IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2017.DOI*

# Advancing Software-Defined Networks: A Survey

**JACOB H. COX, JR.[1], JOAQUIN CHUNG[2], SEAN DONOVAN[2], JARED IVEY [2], RUSSELL J. CLARK [3] (Member, IEEE) GEORGE RILEY[2], AND HENRY L. OWEN, III[2] (Senior Member, IEEE)**

[1]Soar Technology, Inc., Ann Arbor, MI 48105 USA (e-mail: jcox70@gatech.edu)
[2]Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, U.S.A. (e-mail: joaquin.chung@gatech.edu, sdonovan@gatech.edu, j.ivey@gatech.edu, riley@ece.gatech.edu, henry.owen@ece.gatech.edu)
[3]College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332, U.S.A. (e-mail: russ.clark@gatech.edu)

Corresponding author: Jacob H. Cox, JR. (e-mail: jhcox70@gatech.edu).

**ABSTRACT** Having gained momentum from its promise of centralized control over distributed network architectures at bargain costs, Software-defined Networking (SDN) is an ever-increasing topic of research. SDN offers a simplified means to dynamically control multiple simple switches via a single controller program, which contrasts with current network infrastructures where individual network operators manage network devices individually. Already, SDN has realized some extraordinary use cases outside of academia with companies like Google, AT&T, Microsoft, and many others. However, SDN still presents many research and operational challenges for government, industry, and campus networks. Because of these challenges, many SDN solutions have developed in an ad hoc manner that are not easily adopted by other organizations. Hence, this paper seeks to identify some of the many challenges where new and current researchers can still contribute to the advancement of SDN and further hasten its broadening adoption by network operators.

**INDEX TERMS** Software-Defined Networking (SDN), Network Virtualization (NV), Network Functions Virtualization (NFV), Standards, SDN Interfaces and APIs, Data Plane; Middleboxes, SDN Security, Hybrid Networks, Software-defined Exchange (SDX), Software-defined Infrastructure (SDI), Software-defined Wireless Networks (SDWN), Internet of Things (IoT), Information-Centric Networking (ICN), Cloud; Software-defined RAN, 5G

WHILE this paper seeks to cast a wide net over the existing research opportunities for researchers to further advance software-defined networking (SDN) with current and emerging technologies, a caveat remains essential. SDN is a very broad topic, and as this paper demonstrates, it spans many fields. Survey papers covering SDN are equally expansive, in breadth, in depth, and in quantity. Hence, no single paper could possibly provide the scope and depth required to adequately articulate all the ongoing research in this field. Instead, this paper focuses on surveying the proliferation of SDN along with its various applications, while also offering insights as to where new researchers can still contribute and further advance SDN's expansion.

Of course, we do not imply that SDN is being neglected–in fact, it is a highly researched field within both academia and industry, and it has already been adapted by several major IT corporations. For example, according to various reports and research articles [1]–[3], the OpenDayLight (ODL) [1] controller alone has already seen over 100 deployments with companies such as Orange, China Mobile, AT&T, T-Mobile, Comcast, KT Corporation, Telefonica, TeliaSonera, China Telecom, Deutsche Telekom, and Globe Telecom. There are also many drivers pushing network operators to consider SDN. For instance, it is also widely agreed that traditional networks are too complicated to manage, too expensive, and subject to vender-lock-in and ossification [4]. As a result, more organizations are considering SDN as a solution to these problems.

Currently, SDN is best defined as a networking paradigm that decouples the control plane from the data plane, allowing for centralized control of the network along with a global network view where network applications run atop a network operating system (NOS). Additionally, since SDN only deals with packets at the header level, it is primarily utilized with

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



layers 2-4 of the OSI model and other orthogonal protocols, like MPLS. The results of SDN are that it has a logically centralized network intelligence and state and an underlying infrastructure that is abstracted from network applications [5]. Furthermore, these abstractions, sitting atop SDN controllers or network operating systems, make for a powerful concept. Much like a computer operating system can enable high level programming languages by abstracting away the 0s and 1s of binary code, SDN abstracts away the complexities of managing individual network devices. Moreover, proponents of SDN claim several benefits for such configurations, which include 1) centralized control over multi-vendor environments, 2) automation that reduces complexity for users, 3) higher innovation rates, 4) increased security and network reliability, 5) more granular network control, 6) improved user experience, 7) reduced management costs, 8) simplified network deployments, and 9) programmable network services [5].

Already, SDN has been established as a viable network architecture for data centers and backbone networks [6]–[8]. Consider, for example, how Google's B4 project demonstrates SDN's capability to offer a viable NOS with real world applications for traffic engineering and quality of service (QoS) [7]. Other advocates for SDN point to how organizations, such as AT&T, Microsoft, and Verizon, have moved towards SDN to reduce costs, improve utilization, and expedite the delivery of new, client-oriented, services [8]–[10]. Also, as industry is showing greater interest in SDN architectures, so too are government agencies within the United States [11], [12].

However, despite these advantages and use cases, SDN still has limitations that cause concern for network operators who are considering its implementation. As a result, this paper identifies some of the constraints and unsolved problems still facing SDN deployments. For instance, service providers operate networks that are frequently stressed with high traffic volumes, rich media content, and security considerations that SDN does not yet fully address. Another is that SDN still lacks a set of intuitive features that network operators can readily incorporate into their existing infrastructures, which will likely still comprise a mix of legacy hardware devices. Further still are issues with fault-tolerance, universal standards and certification, integration with existing network infrastructure, and much more.

Hence, while the history of SDN [6], [13] and its current state [14], [15] have been thoroughly explored, our work seeks to extrapolate potential research directions, based on needs defined by academia, industry, and governmental organizations, which will help advance SDN as a viable alternative to traditional networks for network operators in coming years. Moreover, unlike many SDN-based survey papers already in existence, we seek to identify open research topics as a means of advancing SDN for stake holders who are either already deploying an SDN or considering the deployment of one. This effort comprises the major contribution of our work.

Our specific contributions can be summarized as follows.

- **Snapshot of SDN Deployment**: Based on current market reports and our own inquiries, we provide a current snapshot for the state of SDN deployments in industry, government, and campus networks.
- **SDN Benefits, Challenges, and Opportunities**: We consolidate information from multiple surveys and white papers, as well as our own research, to provide a list of benefits, challenges, and opportunities as seen by organizations and network operators who are currently deploying or considering deployment of SDN.
- **Research Opportunities**. We identify research contributions still needed to remove barriers for organizations and their network operators considering the integration of SDN into their current architectures.
- **Simulation Environments**. We offer a detailed discussion, including strengths and weaknesses of available simulation platforms that researchers and network operators may consider for testing their solutions or supporting change management objectives.
- **Hybrid SDN Solutions** Traditional networks will not disappear for years (or decades) to come. As a result, we discuss where researchers can find opportunities to assist network operators with partial SDN deployments within their traditional network infrastructure.
- **Next Generation Technologies**. We evaluate next generation technologies (e.g., IoT, ICN, Cloud, and 5G) and identify potential areas to advance these technologies through SDN.

The rest of this paper is outlined as follows. In §I, we discuss the background, current state, recent trends, and research opportunities pertaining to SDN. Additionally, we identify several factors that are key to much broader deployments of SDNs. Having identified key network operator requirements, we next pursue a series of sections highlighting where SDN can be strengthened to further advance its use. Accordingly, we discuss standards, certification, and training in §II. Then, we discuss the challenges and research opportunities affecting the control plane of SDN architectures in §III. Next, we look at the SDN data plane and identify some of the issues that are created with generic, *white box*, switches in §IV, along with the need for additional data plane functionality for which SDN did not originally consider. The discussion then leads into middlebox deployments and utilization in §V. Afterwards, we discuss how virtualization technologies serve as key enablers for advancing the SDN paradigm, including Network Virtualization, Network Functions Virtualization, and other recent developments in §VI. We then discuss the importance of security, both for the SDN infrastructure and traditional network threats, highlighting areas where SDN is suited to assist network operators and where researchers can better enhance it, in §VII. The need for tools to measure performance and validate new SDN applications are discussed in §VIII. Likewise, modeling and simulation platforms for developing frameworks and validating use cases are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



discussed in §IX. Next, hybrid networks are discussed in §X and software-defined infrastructures/exchanges are explored in §XI. Finally, we discuss the roll of SDN and virtualization in emerging technologies (e.g., IoT, ICN, SDRAN, SDWN, and 5G) in §XII before a final discussion of our presented topics in §XIII and concluding in §XIV.

## I. STATE OF SDN

SDN has received a considerable amount of attention from academia, industry, and government in recent years. This interest has even resulted in various levels of involvement and/or deployment by a variety of organizations as indicated in Table 1 and Table 2. Even so, the concept of a programmable network, having a decoupled control plane, has existed for many years–a thorough history of which can be found in [2], [13], [15]. Still, SDN has only recently gained the momentum needed to establish itself as a significant factor within next-generation networking initiatives, which are predicted to exceed $105B per annum by 2020 [16]. Partly, this momentum is due to SDN's flexibility, abstractions, and global network views; however, SDN is also aided by the symbiotic inclusion of virtualization technologies, as we will later discuss in §VI.

1) Dynamic capacity to meet application needs
2) Bringing new services to market faster
3) OPEX reduction
4) CAPEX reduction

Regrettably, one notably absent benefit from the above list is network security. In fact, less than 4% of SDN initiatives undertaken by surveyed enterprises were conducted by security leads, while 73% were led by engineering, DevOps, applications, or operations leads. However, other studies indicate that respondents do see security as a benefit of SDN. For instance, another study [3], which surveyed 201 technology vendors and end-users, found that respondents most appreciated the following benefits and capabilities of SDN.

1) Management flexibility
2) OPEX reduction
3) CAPEX reduction
4) Programmability
5) Monitoring and Visualization
6) Security

TABLE 1: SDN 2015 Adoption Plans by Campus, Industry and Government

| Network | Deploy (Yes) | Deploy (No) | Evaluating |
|---|---|---|---|
| **Industry [17]** | **12%** | **56%** | **32%** |
| **Data Center [18]** | **17%** | **49%** | **34%** |
| **Enterprise [19]** | **72%** | **28%** | **–** |
| **Government [20]** | **37%** | **29%** | **34%** |

TABLE 2: University Engagement with SDN among Leading US National Universities

| SDN Engagement for Top 104 National Universities | SDN Engagement (Yes) | | | SDN Engagement (No) |
|---|---|---|---|---|
| | Teaching, Researching | Deployed (Yes) | Deployed (No) | |
| **Universities** | **84 (81%)** | **46 (44%)** | **38 (37%)** | **20 (19%)** |

Academia has also embraced SDN. In our own research, we canvassed the top 104 national schools within the United States as determined by a national ranking organization[1]. During our canvass, we considered a university to be actively engaged in SDN research if since 2014 it generated one publication relating to SDN, taught SDN as a portion of a course, or participated in an SDN related project. Our canvass, completed in April 2017, determined that roughly 81% of these universities are actively advancing SDN as a legitimate networking paradigm. Additionally, it identified what percentage of these schools have deployed an SDN on their campus–partial or otherwise. That value being 40%. However, this value is somewhat conservative, since universities that did not respond to our queries or indicated uncertainty about an SDN deployment (13 in all) were grouped with universities that indicated no SDN deployment. We provide these results in Table 2.

However, campus adopters of SDN also reported some concerns [21], which we list in descending order below.

1) Cost (41%)
2) Integration with existing systems (38%)
3) Security (36%)
4) Lack of employee skill sets (30%)

While slower to embrace SDN, the United States government has also made recent strides in SDN, with statements regarding recent deployments indicating benefits of reduced equipment costs and improved network performance [20]. Meanwhile, military officials see SDN as an operational multiplier within its future networks. In March 2016, a report was released by the Army's Chief Information Officer [22] indicating that future science, technology, IT, and network investment strategies should include SDN. This same report promotes SDN as a way to create more dynamic, manageable, and adaptable networks that are also more agile, cost effective, and capable of being rapidly configured and redeployed as needed. Other SDN benefits for government networks captured in a Juniper survey [21] are listed below.

1) Improved network performance and efficiency (26%)
2) Simplified network operations (19%)
3) Cost saving on operations (13%)
4) Increased agility via automation and orchestration (13%)
5) Improved services (12%)
6) Enable greater security (9%)

Likewise, government organizations prioritize the below list as criteria for adopting SDN solutions [21].

1) High availability and resiliency (30%)
2) Analytics and reporting (23%)

[1]The 'National Universities Rankings' pole from 2016 serves as our sample group for this work http://colleges.usnews.rankingsandreviews.com/best-colleges/rankings/national-universities

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



3) Automation and rapid provisioning (19%)
4) Open source options (12%)
5) Scale (10%)

Still, government adopters of SDN have also cited several challenges [21]. Such challenges include the following.

1) Cost (51%)
2) Integration with existing systems (34%)
3) Lack of employee skill sets (32%)
4) Security (29%)

Accordingly, government IT decision makers still require that SDN be complementary to traditional networks, since predictions beyond five years indicate that prevalent network infrastructures will be a hybrid of both technologies. Hence network operators must be prepared to embrace hybrid networks. Nonetheless, SDN's promise to more closely integrate network fabrics and external cybersecurity platforms potentially offers greater network visibility along with the enforcement of domain-specific real-time policies–even when integrated with traditional network infrastructures. SDN is also rife with many other research opportunities–some of which may even lead to revisions of the SDN architecture, standards, and operation as researchers begin to achieve an optimal set of capabilities and expectations [11].

As it stands, SDN's decoupling of the control plane from the data plane means that network operators can work with a centralized control program instead of numerous, multi-vendor, network devices to implement their desired policies. Furthermore, with the control plane (i.e., the proprietary operating system for the device) removed, many proponents for SDN believe that the cost of producing SDN hardware will be much cheaper than building traditional network devices [3]. In this environment, vendors must now compete to provide hardware solutions in one arena and software applications in another. As a result, there is now greater opportunity for new manufactures or software developers to compete in either market (i.e., *white box* hardware solutions or network software applications).

An additional benefit we see in SDN deployments is the ability to replace, reorganize, or better chain together various middlebox devices within networks (e.g. load balancers, low-level firewalls, intrusion detection prevention systems (IDPS), and other third-party solutions). In some cases, the SDN paradigm can incorporate security features into its network operating system. In other cases, developments in network functions virtualization (NFV) are allowing network operators to create virtual instances of hardware devices and use SDN to ensure traffic is appropriately (and dynamically) routed through these devices. By removing middlebox hardware, network operators can obtain greater control of their network and dynamically respond to the network's needs in ways previously considered too time consuming or cost prohibitive.

Perhaps, SDN's greatest benefit to network operators is the ability to work at a much higher level of abstraction than ever before. Through this abstraction, operators can focus on network applications, such as traffic engineering, security, policy enforcement, etc. Still, these capabilities are not yet easily realized. For instance, when Google developed their software-defined WAN (SD-WAN), which carries over 90% of Google's data, significant engineering was required from building new switches to developing customized software [7]. They also took advantage of several characteristics unique to their network. For instance, Google owned their infrastructure [7] from point-to-point and had complete control of when data was pushed over the network. While the result was a powerful enhancement of Google's back-end WAN, offering 99% utilization [7], it was very much an ad hoc solution, and not necessarily applicable to other network architectures.

Other areas where SDN is falling short involves standards, switch architecture, full middlebox elimination (e.g., stateful and NextGen firewalls, intrusion detection prevention systems (IDPS), MPLS, Encryption, etc.), standardized interfaces (particularly the northbound interface and the east-west bound interface), security, use cases for integration with legacy systems, clear and intuitive programming languages and APIs, and integration techniques for network applications are just a few. What role SDN will play in conjunction with virtualization technologies, like network function virtualization (NFV), to enhance wireless networks, Internet of Things (IoT), and fifth generation (5G) technologies is not fully understood yet either. Likewise, quality of service (QoS), reliability, scalability, service management, and extensibility are all underdeveloped areas of research in SDN, while programmability and performance issues must be resolved if 100 Gb/s and greater speeds are to be handled by SDN [14].

### A. RECENT TRENDS

While evaluating the state of SDN, it is also important to consider the trends that are driving network operators to consider it. These trends include greater user mobility, an explosion of devices generating unprecedented amounts of traffic, growth of cloud services along with a convergence of compute, storage, and network architecture, and more prevalent virtualization technologies [2]. Accompanying these trends are significant challenges, such as huge capital investments, shrinking gaps between investment and revenues, growing numbers of proprietary and unique hardware appliances on operator networks, shorter hardware lifecycles, lack of flexibility and agility, and unacceptable time to market margins for new services [2]. Furthermore, current trends for picking controllers seems to lie with the momentum behind the controller. For instance, lacking clear standards, network operators are turning to technology they believe will have long-term support and a wide margin of public contributions and use cases [3].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



TABLE 3: Network Operator Requirements for SDN

| Requirements Index | Sections |
|---|---|
| **Training & Certification** | §II, §XIII |
| **Legacy Network Compatibility** | §II, §III, §VII, §X, §XIII |
| **Network Management** | §III, §VI, §XII |
| **Middlebox Management** | §IV, §V, §VI, §VII, §IX, §XI, §XIII |
| **CAPEX Reduction** | §V, §VI, §X, §XII, §XIII |
| **OPEX Reduction** | §V, §VI, §XIII |
| **Policy Enforcement** | §II, §III, §V, §VI, §VII, §XI, §XII, §XIII |
| **Network Monitoring** | §III, §VI, §IX, §XII |
| **Network Security** | §II, §III, §IV, §VI, §VII, §VIII, §IX, §XI, §XII, §XIII |
| **Change Control** | §VI, §VIII, §IX, §XIII |
| **Network Visualization** | §III, §IX |
| **Intuitive Interfaces or APIs** | §III, §XIII |
| **Network Flexibility** | §III, §V, §VI, §IX, §XII, §XIII |
| **Fault Tolerance** | §VIII, §XI |
| **Quality of Service** | §III, §VI, §XII, §XIII |

### *B. SUMMARY OF CHALLENGES AND RESEARCH OPPORTUNITIES*

SDN offers the unique ability to maintain a global view of its network down to port connections, matching on multiple header fields. While doing so, SDN controllers are able to query switch statistics, detect flow patterns of concern, and react dynamically to perceived threats, which also offers numerous possibilities for SDN environments. Because of this, many initiatives are coupled with SDN research. However, SDN must still progress in numerous research areas as we will next discuss before such initiatives will be fully deployable. Of course, SDN research initiatives are also being driven by political, strategic, and cost related decisions that will ultimately drive these areas to change at a fast pace. Hence, we merely attempt to introduce these various research areas and suggest ways that new researchers can currently begin contributing to this growing body of literature and further advance the SDN paradigm.

Researchers must also maintain awareness that network operators make their living by providing services–SDN is merely a platform to provide those services. Thus, operators do not necessarily want an SDN [2]. Rather, what they want is to solve current problems and add value for their clients. So, as long as SDN can allow network operators to either offer or enhance these services, SDN will gain wider adoption [2]. From our observations on the state of SDN, we have identified several areas (or requirements) where SDN can be improved to address network operator concerns. Specifically, we present them in Table 3, where we also identify the corresponding sections where these requirements are addressed.

## II. FEDERATION OF STANDARDS

Introducing a new networking paradigm, such as SDN, requires standardization if it is to be universally adopted. These standards are required for its engineering, integration, operation, and maintenance (E&I, O&M). Likewise, policies must be developed for how SDN domains interact with each other, other legacy domains, and virtualized components. Many telecom executives have said that standards are critical to promoting SDN architectures [9]. Yet, these standards must also consider researchers and network operators to lessen the ad hoc nature of SDN deployments. With such measures, consistent routing, security, and daily network activities can be better ensured.

With regard to the virtualization of core network components, the European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) Industry Specification Group [23] are researching best practices for the efficient and scalable placement of core network components [14]. Accordingly, ETSI has steadily put forward draft standards that have been quickly adopted by various vendors [16]. However, while ETSI has offered VNF and network service descriptors as template definitions of functions and services, it has yet to define a data model for realizing these descriptors [24]. As a result, a lack of clear interfaces for chaining different functions into a singular service remains an open research challenge as we will discuss in §VI-B.

Interfaces, mechanisms and protocols for separating the control plane from the forwarding plane are another aspect of SDN that the Internet Engineering Task Force (IETF) is attempting to resolve within the context of IP routers. Additionally, the Open Platform for NFV Project (OPNFV) was introduced in 2014 to ease industry mobilization around new NFV products and services [16]. For instance, there is still a need to standardize APIs that address interconnect policies, traffic flow management, authentication, and authorization between other prioritized network elements for the successful adoption and advancement of SDN and NFV technologies [2]. As pointed out in other literature [2], [25], [26], the PCRF (Policy and Charging Rules Function), which is part of the Evolved Packet Core (EPC) supporting service data flow detection, could serve as a potential means for how controllers reach consensus for setting up and managing flows. However, while PCRF operates as a policy enforcement entity at the service/application level (including LTE networks), the SDN controller operates at the L2-L4 level. As of yet, no standardized interface exists between the SDN controller and PCRF, and this represents a significant challenge for developers if PCRF is indeed to be used for augmenting SDN controller decisions.

How SDNs will handle such policies is yet another field needing standardization. Some researchers have developed policy engines for Software-defined Exchanges [27] and Security features [28]; however, the SDN community is far from achieving a universal standard that will allow multiple network operators from multiple domains to systematically dictate their policies in a way that an overarching SDN

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



controller can interpret and render while resolving potential policy conflicts.

Regarding certification, the current nature of SDN means that traditional network operators will require retraining and or certification to effectively engineer and integrate SDN architectures, as well as to operate and maintain them. To a certain extent, this new paradigm requires that they be software engineers first, and network operators second[2]. Already, the ONF has established a skills certification exam, ONF-Certified SDN Engineer (OCSE-111) to certify that network operators possess required skills for operating SDNs [29]. Currently, this exam focuses on seven domains as listed below:

1) SDN Concepts
2) OpenFlow
3) SDN Architecture and Ecosystem
4) SDN Implementations and Migrations
5) Troubleshooting and Testing
6) SDN Security
7) SDN Futures

Additional research by independent sources is likely needed to validate whether this certification represents an adequate range of SDN topics for aspiring SDN operators. Nevertheless, this certification represents a starting point for organizations seeking to validate and/or certify the skills of network operators responsible for handling their organization's SDN infrastructure.

## III. CONTROL PLANE

Programmability is the fundamental paradigm shift offered by SDN [30]. It allows for an entire network of individual network devices (once programmed individually) to be programmed or orchestrated as one single entity. In other words, the SDN controller, along with its interfaces, constitutes the control plane, which serves as a logically centralized intelligence having a global view of the network and the ability to dynamically reconfigure it. Said differently, network management has everything to do with the control plane, and this is an extremely powerful abstraction. As a result, this plane (a.k.a. tier or layer) is often referred to as the network operating system (NOS), as it allows the SDN to abstract its physical network state as a global network view to a controller instance running in the application layer [2].

Already, previous publications have attempted to identify key attributes to look for in an SDN controller. For instance, Metzler et al. [31] identify ten things to look for in a controller. These include

1) OpenFlow Support
2) Network Virtualization
3) Network Functionality
4) Scalability
5) Performance
6) Network Programmability

[2]This observation hinges on the development of new APIs for network operators as we will discuss in §III.

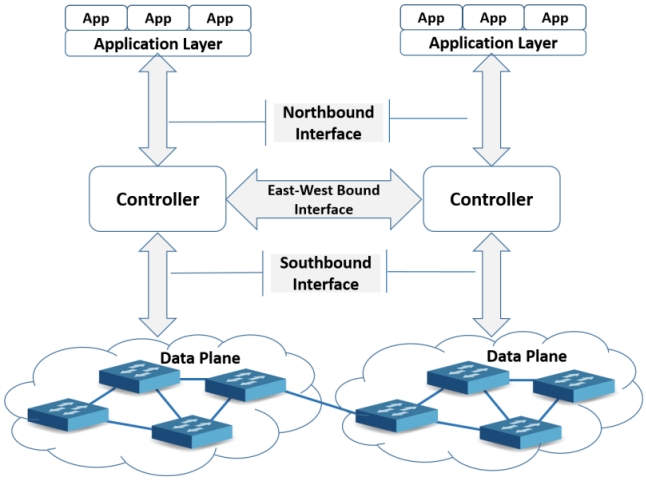



FIGURE 1: SDN Interfaces

7) Reliability [or Fault-Tolerance]
8) Security of the Network
9) Centralized Management and Visualization
10) Controller Vendor

However, as we will explain in §III-C and §XI, there is at least one additional aspect of an SDN controller worth considering (i.e., the east-west bound interface). Likewise, interoperability of controllers must be considered, since significant problems can result if even occasional incompatibility occurs.

Within the control plane are three interfaces or APIs. Those are 1) the southbound interface, 2) the northbound interface, and 3) the east-west bound interface. Technically, these three interfaces represent standards in and of themselves, yet, each of them are at various states of acceptance, development and integration. We depict these interfaces in Fig. 1 with two controllers, both serving as a NOS for their subnetwork. The northbound interface allows control programs (a.k.a. applications) to provide instructions to the controller. Through the southbound interface, the controller sends instructions and receives information from the data plane. The east-west bound interface allows for controllers to share a common network view and to coordinate policy and protocol enforcement. Most of these interfaces currently reside in the open domain (being freely available under open source license), which has proven a significant boon in the development and expansion of SDN. It has also helped network operators to promote vendor-neutral solutions for their organizations.

However, a significant limitation to SDN is that no clear programming language has been identified for its northbound API, and while its southbound API has a de facto standard in OpenFlow, other southbound APIs are competing for attention. Likewise, the east-west bound interface for SDN is ill defined or missing altogether from many controllers. Moreover, SDN's APIs are still in need of more intuitive abstractions that will allow network operators and engineers to

manage how applications interact with the network and better enable network programmability [32]. Just consider Google's B4 project [7], which provides an excellent overview of the challenges of creating an SDN-based back-end network for connecting its data centers. SDN programming, at least at the moment, is still very low level and complicated to write.

Similarly, SDN lacks uniformity across networks. While a Cisco (or Juniper, Brocade, etc.) programmer can move from one organization to the next already having the prerequisite command line interface skills needed to operate on the new network, that is not the case for SDN networks. This is a significant limitation for organizations having high turn-over rates, and it requires significant retraining of new network operators. This limitation makes adaptation of standardized SDN programming languages a critical goal for SDN's continued advancement.

### A. SOUTHBOUND INTERFACE OR API

The southbound API provides the interface between controllers and switches using protocols like OpenFlow [49], [50], ForCES [51] and POF [52] to update flow table entries in switches. Put another way, southbound protocols define the control communications that occur between the controller and data plane devices, which include physical and virtual switches and routers [2]. Yet, while southbound APIs, like OpenFlow, make it possible to program networks, they do not necessarily make it easy [53]. Partly, this is because OpenFlow is closely tied to its underlying hardware. As such, policies are incorporated through bit patterns, and this can make southbound APIs more difficult to intuit. For this reason, the southbound API is used as an interface to enable high-level programs to dynamically interact with the network's physical (or virtual) devices.

It is also important for the southbound interface to accommodate greater flexibility in the control plane so that new control methods can be quickly adopted [54]. Since, southbound APIs can be more difficult to reason about, a stable and common platform is required, and OpenFlow [49] is the most widely used of the southbound APIs in both academia and industry. For this reason, we consider primarily OpenFlow-enabled switches throughout this paper.

In general, the southbound API allows us to match certain fields in packet headers and apply actions accordingly. For instance, OpenFlow v1.3 [55] can match source and destination fields for TCP or UDP ports, IP and MAC addresses, switch ingress and egress ports, MPLS labels, and differentiated services field code points for QoS [55]. With each match, OpenFlow v1.3[3] can set the output port, drop the packet, set the queue or push/pop MPLS labels [55].

For the most part, just as hiding assembly code from application developers allows them to focus on their code's functionality, it makes sense that the southbound interface should be hidden from network operators, so they can better focus on the utilization of SDN's applications on their networks. Hence, it becomes more important that the network operator understand the northbound interface and the applications interacting with it. Still, while OpenFlow is considered the de facto standard, other southbound protocols, like OVSDB, BGP[4], XMPP, and NETCONF have gained sufficient enough popularity that network operators must ensure that the controllers they select will properly interact with the data plane. According to [3], there has recently been much debate over the compatibility and scalability of various controller frameworks having varied support for southbound protocols that include OpenFlow, BGP, PCEP, SNMP, NETCONF, OVSDB, etc. [2]. In Table 4, we list a number of popular controllers and indicate whether they support OpenFlow (and what version). We also indicate, where applicable, what percentage (%) of SDN deployments utilize these controllers. Note also that multiple controller types may be deployed by various respondents. As a result, adding these percentages provides a value greater than 100%.

#### 1) OpenFlow

As the de facto standard for the southbound interface, it is still worth noting that some researchers still do not consider OpenFlow to be a complete standard [2], [56]. Its shortfalls include: 1) the lack of a mechanism for fully managing devices (i.e., for controlling port/trunk interfaces and queuing); 2) poor network status communication (i.e., loads, traffic, and operational state for nodes and trunk); 3) absence of a specific method for establishing communication channels between the controller and switch; and 4) making bare metal switches operational from a newly installed state [2], [56]. Nonetheless, OpenFlow still offers several intellectual contributions, which include: 1) generalizing network devices and functions, 2) the concept of a network operating system, and 3) distributed state management techniques [2], [13].

#### 2) Argument For Two Separate Standards

While not a new argument, previous work has argued that two versions of OpenFlow should be considered: one for edge networks and one for core networks [4]. The authors argue that current OpenFlow versions are too tightly coupled with the host-network interface, meaning that each switch must consider original packet header information for all forwarding decisions. So, OpenFlow may unnecessarily couple host requirements to network core behavior [4]. Hence, current OpenFlow protocols must struggle with finding balance between practicality (support for matching standard headers) and generality (support for matching all headers). As a result, OpenFlow switches must match on hundreds of bits, whereas core forwarding protocols, like multi-protocol label switching (MPLS), matches on only tens of bits. Hence, OpenFlow does not meet the mark for simplified hardware.

With OpenFlow seeking to balance this practicality with generality, one can argue that it is not general enough for the

[3]The most recent version of OpenFlow as of this paper is 1.5.

[4]Border Gateway Protocol (BGP) is a well-known core Internet protocol that is also being adapted for use as a hybrid SDN protocol. See https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



TABLE 4: Controllers

| Controller | Lang | PD | OF | MT | TLS | RA | % | Other Features |
|---|---|---|---|---|---|---|---|---|
| **OpenDayLight [1]** | Java/Python | Y | 1.3 | Y | Y | Y | 61% | GUI |
| **ONOS [33]** | Java | Y | 1.3 | Y | – | – | 23% | Built for Service Providers. Also, supports OVSDB, BGP, Netconf, TL1. |
| **OPNFV [34]** | Java/Python | – | 1.3 | Y | Y | Y | 26% | Supports ODL, ONOS, or OpenContrail, NV/SDN. |
| **DIFANE [35]** | **C** | Y | 1.0 | Y | – | – | – | Built on NOX |
| **HyperFlow [36]** | **C++** | Y | 1.0 | Y | – | – | – | Built on NOX. |
| **NOX [37]** | **C++** | N | 1.0 | N | – | – | – | Deprecated. |
| **SNAC [38]** | **C++** | N | 1.0 | N | – | – | – | Built on NOX, GUI, closed source. |
| **POX [39]** | Python | N | 1.0 | N | – | – | – | Development stagnated, GUI. |
| **Pyretic [40]** | Python | N | 1.0 | N | – | – | – | Built on POX. [Deprecated.] |
| **Ryu [41]** | Python | N | 1.5 | N | Y | – | 15% | Frequent switch certifications. |
| **Ryuretic [42]** | Python | N | 1.5 | N | Y | – | – | Built on Ryu. |
| **Beacon [43]** | Java | N | 1.0 | Y | – | – | – | Limited to STAR topo, GUI. |
| **Floodlight [44]** | Java | N | 1.4 | Y | Y | Y | 11% | Forked from Beacon, GUI. |
| **Trema [45]** | Ruby/C | N | 1.3 | – | – | – | – | |
| **Kandoo [46]** | Go | Y | 1.0 | Y | – | – | – | |
| **OpenContrail [47]** | Java/Python | Y | – | Y | Y | Y | 14% | Developed for OpenStack. Southbound API: XMPP, BGP, and NETCONF. |
| **OpenMUL [48]** | C/Python | Y | 1.4 | Y | Y | Y | 8% | Supports NETCONF & OSVDB. Created for stability and performance. |

Lang. (Language), PD (Physically Distributed), MT (Multi-Threaded), TLS (Transport Layer Security) , RA (Rest API), OF (OpenFlow Version), % (Deployed Percentage [3])–Note some deployments may use more than one controller type.

edge or simple enough for the core. Hence, Casado et al. [4] make a strong argument for two separate OpenFlow protocols, allowing the edge and the core fabric to be logically controlled by separate controllers. In one case, the edge can offer semantically rich services, including network security, isolation, mobility, access control, etc. In the other, the fabric can focus on high-speed packet forwarding with a minimized set of forwarding primitives [4]. In both cases, the control planes can evolve separately.

One reason we choose to revive this argument is that next generation technologies (e.g., IoT, ICN, 5G, and others) will continually push the network's edge to adapt to new and unforeseen requirements. Hence, flexibility is a key enabler for these technologies. Allowing OpenFlow to adapt to edge requirements separate from its core can then serve to further assist efforts to advance the SDN paradigm in support of emerging technologies (see §XII). Furthermore, the existence of edge and core protocols may also create better opportunities for the creation of hybrid SDN networks, as we will discuss in §X.

### B. NORTHBOUND INTERFACE OR API

The northbound interface links control programs (a.k.a. applications) to the controller (or network operating system). Through this API, applications can orchestrate (or program) the data plane to perform complex tasks like traffic engineering, topology discovery, quality of service (QoS), load balancing, security policy enforcement, firewalls, delay and jitter management, and much more [53]. Yet, despite its ability to provide the above features, no clear northbound interface standard yet exists [57]–[59]. What's more, capabilities and features can drastically differ based on the network operator's choice of controller [57]. Hence, intuitive abstractions for network operators to create their own policies, protocols, and applications are still lagging [60], which has led other researchers [61] to further explore the challenges of developing SDN software.

As alluded to above, many northbound APIs exist with varying strengths and weaknesses. However, their capabilities, intuitiveness, documentation, etc. vary broadly across available controllers, which may be a single centralized controller or a distributed controller that is logically centralized. On one hand, NOX [37], Floodlight [44], Maestro [62], Beacon [43], SNAC [38], OpenDayLight [1], and Trema [45] are all network operating systems having a single controller. On the other hand, Onix [63], HyperFlow [36], ONOS [64] and DIFANE [35] are examples of a distributed controller. Additional information about these controllers and their available features are presented in Table 4.

Another challenging aspect of the northbound interface is that while available APIs attempt to offer abstractions, they still fail at being intuitive [65]. For instance, most northbound APIs require a deeper knowledge of the programming constructs that comprise them. For researchers, this can be challenging. For network operators, who have limited resources and multiple network responsibilities, the time and effort required to implement SDN protocols and policies on their networks may prove too much of a barrier [65]. In other words, network operators may lack a sufficient understanding of the northbound API and be too busy with

immediate network issues to risk the time needed to modify SDN applications and/or potentially introduce new errors to their network. For this reason, we argue that abstraction without intuition is useless. If standardized and intuitive northbound APIs can be developed, then this can further ease the transition of SDNs into existing infrastructures.

Unfortunately, many if not most northbound APIs are poorly documented (or still in development), making it difficult to standardize their use across network infrastructures. Additionally, there is no current means to certify network operators with any of these interfaces. As a result, the northbound API, in its current state, fails to offer network operators a means to create applications in a systematic and intuitive format that is easily interpreted by other coworkers or future successors [60]. Hence, having developers include common APIs like the REST API in conjunction with their own set of controller APIs would go a long way towards providing a standard interface for multiple controllers.

For network operators, the above issues are a huge problem, since the current APIs only provide for ad hoc or custom solutions that do not necessarily translate to other network architectures or services. A natural approach has been to add high-level domain specific languages such as Pyretic [40], NetKAT [66], NetCore [67], Ryuretic [42], NetEgg [60], [65], etc. to allow network operators to program their own policies, protocols, and applications. These programming languages raise the level of abstraction beyond the controllers' APIs and reduce code sizes by orders of magnitude [60]. Nevertheless, while network operators are experts at configuring switch and router solutions, they may not be prepared to program them using domain specific, object oriented, languages [60]. For network engineers and researchers looking to implement specialized services or custom solutions for network architectures, the northbound interface provides an option, but at a cost of time and labor hours. Consequently, the northbound interface is primarily benefiting researchers, software engineers, and corporations with deep research budgets. Until an intuitive and standardized northbound interface is established, SDN will continue to struggle with gaining momentum in current infrastructures maintained by network operators who lack sufficient programming backgrounds.

Still, other researchers have already recognized the need for a hypervisor to incorporate "best of breed" controller solutions so that network operators may run any combination of controller applications in order to meet needs specific to their networks [68]. CoVisor basically allows multiple and disparate controllers to manage shared traffic collaboratively. Additionally, beyond allowing an operator to interact with disparate controllers, like Java-based Floodlight [44] and Python-based Ryu [41], it further allows network operators to compose data policies in three different ways: parallel, sequential, and overriding. Parallel actions allow multiple controllers to simultaneously and independently act on identical packets while sequential policies allow one controller to process packets prior to the next controller [68]. In essence, sequential operation can be seen as a packet having to pass through a firewall application before reaching a load balancing/routing application. The third action, override, gives one controller priority to act or defer on arriving packets [68]. The implication for platforms such as CoVisor is that, provided its interface can be standardized, it would allow network operators to work with one interface and orchestrate multiple controllers as CoVisor's hypervisor captures and processes industry standard OpenFlow messages–while also modifying them to match operator-specified policies using their best features. In this vain, Yuan et al. [60] developed a controller agnostic, programming framework, called NetEgg, that works with timing diagrams and topology examples developed by network operators to produce implementations from the provided examples. Future development as of this paper is still ongoing.

The Northbound Interface (NBI) Working group of the ONF [69] has started working towards an intent-based networking interface for the SDN controller. Under the intent-based networking paradigm, the network operator will describe *what is needed*, as opposed to *how to do it*. For instance, an intent could be "Latency lower than 150 ms for voice traffic"; while the prescription could be "assign all IP phones to the Voice VLAN, mark all traffic from/to the IP Phones as high priority, and put the traffic in the priority queue". Some examples of intent-based networking proposals are the ONOS intent framework [64] and NetAssay [70], an intentional network monitoring framework. Recently, the ONF introduced their open source northbound interface (the Boulder project) for intent-based networking. Boulder chooses an initial information model and the architecture base to form an intent-based interface to an SDN controller (e.g., OpenDayLight or ONOS) [71].

As programming languages are developed, it is critical that developers realize that abstractions without intuition will be worthless to the network operator. Equally important to network operator intuition is the consideration of the over-head such abstractions create and possible performance degradation [30]. However, other work such as LegoSDN [72], which is intended to introduce fault-tolerance to SDN architectures, may also point to a future where applications run in sandboxed VMs communicating with the controller through remote procedure calls. The result could provide better abstractions and greater flexibility for application development.

Applications are generally discussed in terms of a top-level tier (see Fig. 1) where the control and data planes make up the subsequent tiers [2]. We also acknowledge that applications communicate with the controller via its northbound interface or API. Generally speaking, a controller by itself is only as useful as the applications that are available to it [3]. Perhaps the greatest hindrance to a wider SDN deployment is the lack of applications available to network operators. While there are immediate advantages offered by SDN to network operators (e.g. IPv6, traffic engineering, network virtualization, load balancing, security, access control, network monitoring

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



and analysis, network service discovery, etc.), most of these features must be programmed into the network operating system. Yet, many of these functions could be provided to network operators as functions that are either turned on or off. One can imagine a selection list where IPv6 is simply enabled in a graphical user interface or purchased through an app store (e.g. HP [73]), which ultimately instructs the network operating system to complete the setup. Hence, we observe that network tools (e.g., network monitoring, management, and analysis tools) or applications are still required to enable network operators to visualize the deployment of SDN networks [32].

SDN controllers will almost certainly run numerous apps providing different functionality. That is to say that the control logic for various apps will run as separate processes on the controller hardware inside each domain [2]. Examples range from current network protocols and services to tailored apps capable of exploiting network visibility [32]. For instance, Hock et al. [74] have developed an SDN graphical user interface (GUI), called POCO-PLC, which allows network operators to investigate loads and latencies on controllers at different placements during and SDN deployment. The POCO-toolset, which is available online [75], also provides real-time comparisons across multiple controllers.

Network monitoring is yet another service desired by network operators. SDN should allow them to submit network queries based on the user, application, and device that is initiating and receiving services, along with its associated network or domain, and possibly the path its traffic takes [70]. Such an application can assist network operators with monitoring application usage and properties of network paths utilized per user or device and aid with billing services or troubleshooting [70]. NetAssay [70] represents a first attempt at providing intentional network monitoring of domain names and autonomous systems for network operators, yet it is far from complete. For instance, dynamic network conditions impose significant research challenges since autonomous systems frequently announce new prefixes and domain names, incorporate new IP address, and continually join and drop users from the network [70]. Dealing with these dynamic events and maintaining a consistent mapping of monitored services has yet to be solved.

How network information is conveyed to the network operator is equally important. Applications for graphical user interfaces that interact with SDN controllers to build network diagrams and render performance, availability, and utilization depictions as well as network and security alerts are crucial to operation and maintenance support.

Beyond these applications is the need to allow them to share state without sharing fate. For instance, as we will discuss in §VIII-B, the fates of applications and the controller(s) are frequently tied together. As a result, if one fails, so do they all.

### C. EAST-WEST BOUND INTERFACE OR API

How controllers interact with one another to share information within an SDN is handled via their east-west interface. This interface can also be utilized for to server-to-server or server-to-controller communication. Essentially, we can think of this interface as a conduit passing through various network domains to communicate with their subsequent control planes [30]. In doing so, controllers can pass network state information and influence routing decisions. Furthermore, this interface can be used to enhance intra-domain and inter-domain communication, and improve scalability and interoperability of SDN deployments [54]. From the network operator's perspective, this interface should be essentially seamless to setup, but still capable of supporting protocols like BGP [30].

This communication channel is extremely important for large networks (e.g., enterprise, Internet exchanges, Software-defined exchanges, and other multi-domain networks). For example, when large networks are partitioned into multiple, smaller networks, it is likely that network operators will consider including a dedicated network operating system (NOS) for each network. However, a global network view is still required to successfully route packets through the network. This global construct requires that an east-west bound interface exist and provide full mesh connectivity so that heterogeneous network operating systems can exchange network views or coordinate packet routes [76]. In this global design, each individual network is referred to as a subnetwork, and each subnetwork runs its own NOS or controller (See Figure 1). Yet, the subnetwork NOS only has a global view of its subnetwork, which includes topology, reachability, network protocols, network state, entities, etc. Since network operators require a global view of their entire network, it is imperative that local controllers be able to communicate with each other and share network views through an east-west bound interface.

An east-west bound interface is also important for researchers seeking to automate network decisions in order to limit network operator involvement with the network. For instance, network operators may wish to incorporate a Trusted Agent [77] to facilitate security policy transitions within a network by updating or removing policy enforcements. Without an east-west bound interface, the communication protocols between a Trusted Agent and the SDN controller is limited. Additionally, while east-west bound interfaces do exist for some controllers (e.g., Onix [63], HyperFlow [36], DIFANE [35], etc.), their communications are generally private and unreadable between heterogeneous controllers [76]. This represents another challenge for advancing SDN, and translation modules must be considered for SDN controllers to properly interact with legacy equipment if hybrid (see §X) networks are to be fully realized [30].

### D. DISTRIBUTED CONTROL PLANE

For the purposes of scalability, availability, and robustness, general consensus calls for the control plane to be physically

distributed while offering logical centralization [78]. Such measures ensure that another controller is available to assume network control in case of a controller failure. Yet, distributed control plane design is arguably still one of the key challenges in software-defined networking [33], [63], [78], [79].

Issues that arise in a physically distributed control plane include delay of control communications, spoofed control messages, inconsistent updates, and network routing changes–to name just a few–while packets are still in transit. For instance, when incorporating redundancy models, poor filtering of late commands, caused by network delays, may introduce incorrect updates to the data plane. Additionally, the multicasting of every event to all other controllers can create a performance hindering overhead. Malicious controllers can also impact the control plane's network view by not acting as expected [78]. One method for defeating malicious controllers includes the use of cryptographic signatures to indicate whether other controllers agree with a proposed update. Still, even this method is vulnerable when information for the controller is delayed (or prevented) or when elements of the network are subjected to replay attacks.

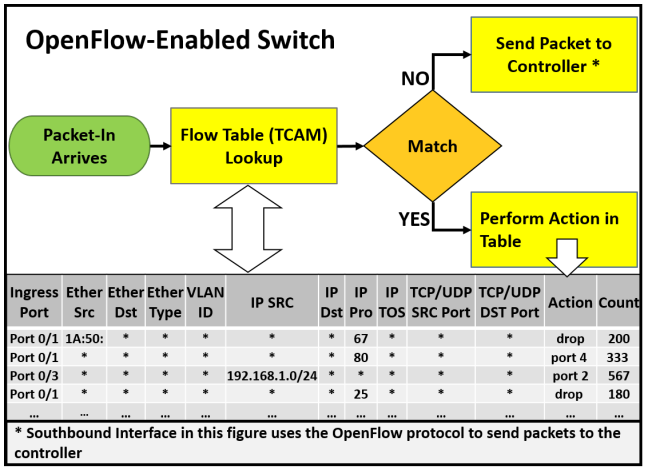
A more recent work by Schiff and Schmid [78], relies on tracking the network's state within the network element (i.e. OpenFlow switch) of a data plane. When controllers join the network, they receive history and state information from the switch and validate it via an XOR hash. Their work, however, requires modifications to the OpenFlow protocol, and they argue that wide agreement already exists for latency critical functions, such as fast failover, be implemented in-band. Additionally, they observe a trend towards putting more state in the data plane (e.g., OpenFlow 2.0/P4 [80]). These challenges represent future research as discussed in §IV. Other work such as Fleet, [81], deals with the malicious administrator problem by assuming a redundantly managed network by multiple administrators or SDN controllers. However, Fleet still cannot handle incomplete and delayed information.

As of this work, the distributed control plane architecture, while needed to support scalability, availability, and robustness, is still vulnerable to attacks focusing on the information handled by its controllers. Likewise, the joining and leaving of controllers is also still not fully supported in distributed architectures. More research is yet required to adequately address these problems.

## IV. DATA PLANE

The data plane or forwarding plane is generally responsible for ensuring the proper transit of traffic from one ingress interface (i.e., an input port) to an appropriate egress interface (i.e., an output port). Primarily, this transit follows the match:action rules contained in network device forwarding tables or forwarding information base (FIB). This data might also be stored in ternary content-addressable memory (TCAM) along with associated metadata, such as packet, flow, and port counters [2].

One assumption of the early SDN research community was that network routers would be simple and homogeneous,




| Ingress Port | Ether Src | Ether Dst | Ether Type | VLAN ID | IP SRC | IP Dst | IP Pro | IP TOS | TCP/UDP SRC Port | TCP/UDP DST Port | Action | Count |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Port 0/1 | 1A:50: | * | * | * | * | * | 67 | * | * | * | drop | 200 |
| Port 0/1 | * | * | * | * | * | * | 80 | * | * | * | port 4 | 333 |
| Port 0/3 | * | * | * | * | 192.168.1.0/24 | * | * | * | * | * | port 2 | 567 |
| Port 0/1 | * | * | * | * | * | * | 25 | * | * | * | drop | 180 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

* Southbound Interface in this figure uses the OpenFlow protocol to send packets to the controller

FIGURE 2: OpenFlow-enabled Switch

working with IPv4 forwarding and Ethernet MAC switching [82]. It was also assumed that simple Ternary Content Addressable Memory (TCAM) forwarding would be used. As a result, early OpenFlow [50] switches were designed with simple lookup tables (i.e., Flow Tables) to perform header matching and then execute various actions as specified in the flow table (match:action). An example of an OpenFlow-enabled switch is shown in Figure 2. In this diagram, the header fields of arriving packets are matched against header fields located in the TCAM or flow table. The action column indicates what to do with the packet if a match is found. If no match is found, then the switch forwards the packet to the OpenFlow controller using its southbound interface, as previously described in §III, to obtain new flow rules. It suffices to understand that the southbound interface refers to the protocol used by data plane elements (e.g., switches) to communicate with their controller.

In truth, network operators deploy routers that support multiple kinds of forwarding protocols, like MPLS, carrier Ethernet, IPv4 & IPv6, etc. This realization has made protocol-independent processing a leading goal of SDN [80], [83], and some SDN researchers now seek to enable the processing of packets independent of implemented control protocols. As stated above, early researchers also assumed simple TCAM forwarding, yet TCAMs are expensive, power-intensive, and not very scalable. With such limitations, *white boxes* (i.e., SDN switches) still have issues with integration, interoperability, and performance. For instance, security applications, like NetSight [84], require firmware and hardware modifications to handle compression. Also, SDN applications are not portable from one vendor platform to another without added modification. Thus, OpenFlow is still hardware dependent [53], and SDN platforms are still not truly vendor-agnostic. Such limitations must be addressed if network operators are to see SDN as a vendor neutral solution for their networks.

One project aimed at addressing cross-platform portability

is the Indigo project, which is an open source initiative within the Floodlight project [44]. Its goal is to offer software that supports thin switching with stable and interoperable implementation for the OpenFlow [50] protocol. At least one company has embraced this initiative by developing thin switching platforms for bare metal switches and virtual switches [85]. The current platform supports OpenFlow v1.3 and programs the rules and policies into forwarding tables for both ASIC-based physical switches and Open vSwitch virtual switches [85].

To help free the SDN control plane from its current hardware constraints (and limitations of OpenFlow), other research now seeks to capitalize on opportunities within the data plane by developing switches containing programmable hardware [53]. These programmable data planes seek to merge the flexibility of software with the performance of hardware. Current research [86] proposes a Reconfigurable Match Table (RMT) architecture, which offers a programmable data plane and protocol-independent processing (including protocol-oblivious forwarding). As it happens, the desired modules required to implement a majority of the features that interest network programmers are fairly limited [53]. This realization has led researchers to believe that a flexible data plane can be achieved through a fixed set of modules [53]. In turn, these modules serve as building blocks, which allow researchers to build fast and programmable data planes.

Of course, such designs require that match tables on switches (e.g., Ternary Content Addressable Memory (TCAM) and Static Random-Access Memory (SRAM)), be flexible [53]. Unfortunately, TCAMs are already expensive and power-hungry, and additional flexibility may make them more so. To overcome TCAM's capacity limitations, some researchers propose adding a network processor within the data plane of hardware switches or utilizing a software agent on hardware switches to assist with scalability issues [87].

In contrast to current capabilities, network operators may also desire to perform more complex operations within the data plane (e.g., encryption, deep packet inspection, and real-time transcoding) [53]. Consequently, these requirements mandate that more sophisticated packet processing modules be placed in the data plane–if middleboxes are to be excluded. Realizing this challenge, researchers have advocated for the placement of Field Programmable Gate Arrays (FPGA) or RMTs in *white box* switches [80]. By doing so, a high-level language, like P4 [88], can be used to compile code to FPGAs located on data plane switches and reconfigure network hardware as needed [53]. This concept achieves protocol and target independence and allows for network engineers to reconfigure how their switches process packets after deployment. For languages like P4, other assembly languages (e.g., NetASM) have emerged to directly affect underlying device capabilities and allow network operators to specify their hardware layout on multiple and varied targets with a precise, or fine-grained, level of control [89].

Other research [83], however, has observed that software switches, like Open vSwitch [90], are playing an important role in data centers. Likewise, they observe that nearly every packet in route to or from a virtual machine (VM) passes through a software switch. Moreover, Open vSwitch is a multilayer, production quality, virtual switch that is designed to enable large-scale network automation while supporting standard management interfaces (or APIs) and protocols such as NetFlow, CLI, sFlow, IPFIX, RSPAN, LACP, and 802.lag.

Whether these new chipsets include additional hardware (e.g., CPU/GPP, NPU/NFP, PLD/FPGA/ASSP, FlexPipe, etc.), it is now evident that a new set of assembly languages are needed to compile the various network programming languages (i.e., Flowlog, NetKat, OpenState, OpenFlow, P4) to hardware [14]. As a result, one goal for SDN may be to develop networks from general purpose hardware with the realization that the end model will likely be a programmable hybrid of current architectures [14]. This also means that network operators must be made aware of these capabilities and provided abstractions to modify these boxes in a way that adheres to the SDN paradigm of network programming. Otherwise, some network functions must be included as middleboxes within the data plane as well, which may be more likely.

## V. MIDDLEBOXES

Network Functions (NF) or middleboxes comprise 40%-60% of devices utilized in large-scale networks [91]. Generally speaking, middleboxes represent stateful systems that are purpose-built to support narrowly defined and highly specialized network functions. As a result of these devices, network operators must think beyond the routing infrastructure to also consider middleboxes, which implement an entire gambit of functions (e.g., firewalls, deep packet inspection, load balancing, IDS, IPS, WAN optimization, proxies, gateways, transponders, encryption devices, VPN, policy enforcement, and other devices.).

Unfortunately, the fixed placement of these devices frequently contributes to network ossification. That is to say that network function hardware makes it difficult to dynamically alter network routes without circumventing the functions provided by this hardware. Middleboxes also represent significant capital and operational expenditure, which includes purchase, management, and replacement costs. Consequently, it is the inclusion of these devices that makes the networking landscape far more complex than SDN researchers originally considered [82].

At this point, it is still difficult to determine which network functions are better handled by SDN architectures. For instance, the networking community has yet to categorize which network functions are best handled by SDN, NFV, hardware, or other technology. Therefore, a greater understanding of SDN and network function capabilities is needed to determine which functions can or should be assumed by SDN (e.g., load balancing and traffic engineering) and which should remain on middlebox platforms (e.g., encryption and deep packet inspection). Doing so will allow network op-

erators to better assess their own networks and determine the overall financial and operational benefits of using SDN architectures and its applications. Additionally, various network function implementations must now be considered by network operators. Those include middleboxes that are implemented both in hardware and virtually and now software-defined network functions.

Consequently, a safe and programmable framework for network function orchestration that can unify a vast set of network functions could greatly assist network operators [13]. In this regard, SDN not only offers to alter the way we do routing, but also how we implement network functions. In some cases [7], [8], [30], [92], network engineers have utilized the capabilities of SDN to implement load balancing, traffic engineering, and low-level firewalls. However, the simplicity of *white boxes*, as discussed in §IV, also means that they are incapable of implementing features like encryption, web caching, and deep packet inspection.

Network function hardware or middleboxes represent a growing number of proprietary hardware devices, all of which sit between forwarding elements while making decisions about packets passing through them. In fact, Sherry et al. [93] find that the number of middleboxes on a network, regardless of the network's size, are roughly equivalent to the network's number of routers and switches. Additionally, as new services are required, so must new middleboxes be obtained, which also requires the allocation of additional space and power.

Unfortunately, facilities such as data centers, cyber centers, network operating centers (NOCs), Internet exchanges, and Internet service providers all have finite space and power constraints that make it difficult to readily accept new boxes [94]. Additionally, middleboxes often offer poor versatility and flexibility, and they frequently lack a general programming interface [93], [95]. For instance, to address the versatility issues of middleboxes, network operators may have to chain multiple boxes together to achieve desired outcomes. This requirement, however, can impose a cost of latency, throughput, and path restrictions. Other concerns for network operators involve replacement costs for devices, which are highly specialized, proprietary, and expensive, and shrinking life-cycles caused by device manufactures attempting to keep pace with technological innovations. Of equal concern is the availability of skills needed to engineer, integrate, design, and operate these devices [94]. Taken as a whole, middleboxes, while useful, are extremely disruptive to network operators seeking flexible and dynamic network capabilities.

Consequently, when a network solution needs to be integrated with more mature, feature-rich, third-party solutions, simplifying service chain deployments is an important factor [3]. Hence, in some cases [96], [97], researchers have deployed SDN to dynamically re-route traffic so that it passes through required network solutions or middleboxes. By doing so, network operators can steer traffic to appropriate middleboxes and avoid the placement of additional middleboxes along other routes in order to avoid the complications of steering traffic via manual CLI. As a result, they hold that middleboxes should be adapted as "cleanly as possible" into the SDN environment [98].

Still, another limitation of SDN, as observed by Fayazbakhsh et al. [98], is that SDN's ability to enforce and verify network-wide policies do not extend to networks with middleboxes, since middlebox usage violates two key SDN tenants [37], [98], [99]. The first is *Origin Binding* where packets are strongly tied to their origins. The second is *Paths Follow Policy* in which case policies are to determine the paths taken by packets. However, they point out that no roadmap currently exists for SDN switches to replace the stateful processing offered by some middleboxes. They also note that significant deployed infrastructure already exists in many enterprises and will not likely go away in the immediate future [98].

As a solution for dealing with middlebox hardware, they introduce FlowTags [98] so that middleboxes insert export tags to provide packets with the necessary causal context needed to restore *Origin Binding* and *Path Follows Policy tenets*. However, the FlowTags solution does come at a cost of adding extension software to middleboxes. Other researchers are calling for an overhaul of SDN that places some intelligence back in the data plane's edge devices in order to keep high-speed, forwarding devices in the core [4], [82]. How this might morph with the incorporation of virtualization technologies is discussed in §VI. Likewise, initiatives supported by ETSI NFV are causing middleboxes to further evolve. Yet, these scenarios still force the network operator to deal with proprietary systems, which require manual configurations, on their networks. Fortunately, as we will discuss in §VI, there exists other platforms (e.g., network functions virtualization) that make it possible to implement more complicated network function solutions. However, until the proprietary functions of all middleboxes can be fully virtualized, methods for integrating middlebox hardware into SDNs are needed.

## VI. VIRTUALIZATION

As SDN has developed, so has its integration with Network Virtualization (NV), Network Functions Virtualization (NFV) and cloud technologies. Additionally, because of the sheer scale of today's data center, virtualization has become much more economic, drawing investment and skills towards generic server technology [100]. Accordingly, as of 2013, nine million generic servers are bought annually in contrast to the mere 180K edge routers [100]. Hence, predictions indicate that network equipment facilities will start to look more and more like data centers, where virtualized network functions are managed in common with other IT management processes.

We will discuss how SDN and virtualization are contributing to emerging technologies in §XII. However, for the remainder of this section, we will discuss these three virtualization technologies in greater detail along with how SDN is being advanced with these platforms. Consequently,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



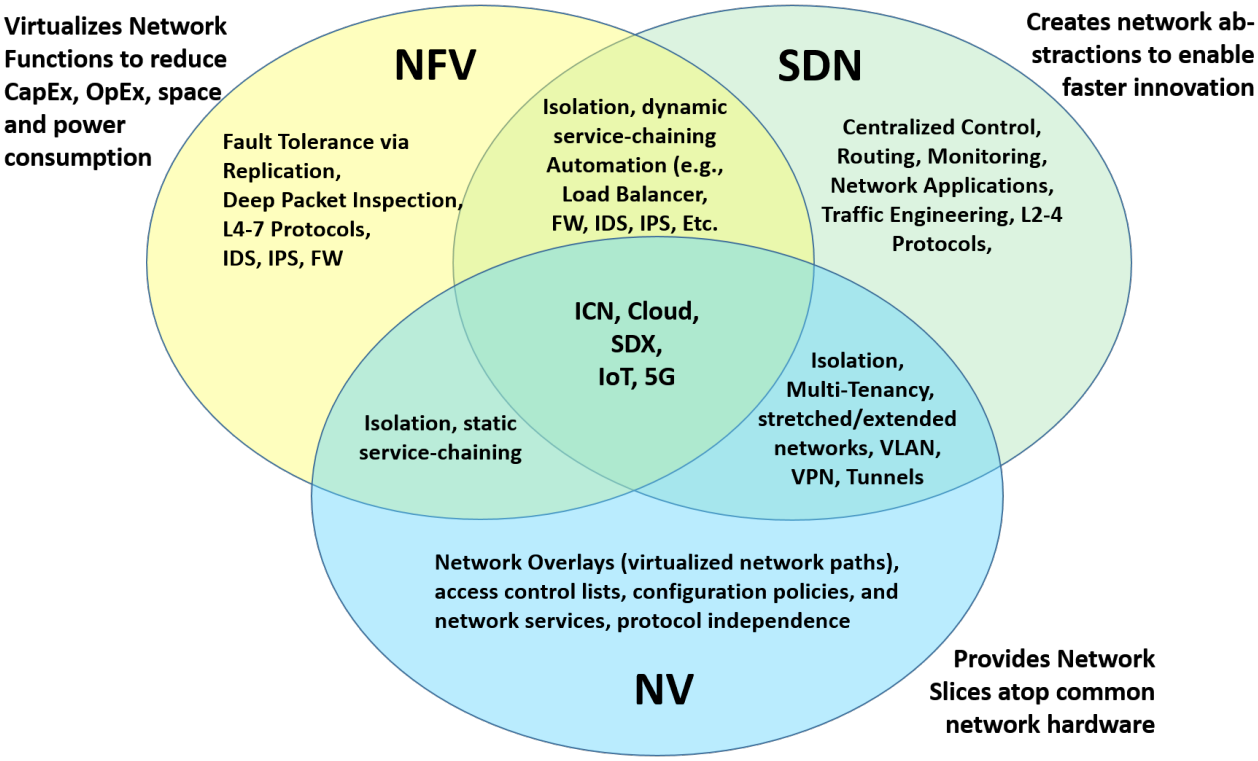



FIGURE 3: Relationship between SDN, NFV, and NV

Fig. 3 provides a Venn diagram of how these technologies overlap with one another.

### A. NETWORK VIRTUALIZATION

Where SDN separates the control plane from the data plane, virtual networks (VNs) divide logical and physical networks. Accordingly, with Network Virtualization (NV), the physical network continues to forward traffic using standard routing protocols as expected; however, the virtual network maintains various overlays, access control lists, network services, and configuration policies to create logical network separations or *slices*. The concept of network virtualization is not new either as it has been previously realized in the form of VLANs and VPNs–both of which are highly successful examples of building separate virtual networks atop a physical infrastructure [2]. However, network virtualization goes beyond VPNs in that it also enables independent programmability of virtual networks [101].

Hence, NV utilizes current network infrastructure to provide multi-tenancy, typically in data center networks, along with traffic and address isolation. Additionally, it is used to realize stretched/extended networks (mobile VMs and dynamic reallocation of resources) [2]. Already, NV is the most common use case for SDN controllers [3]. Many of which appear in data centers, which primarily use two approaches for NV deployment [3]. The first is to directly program the fabric to provide a hop-by-hop virtualization [2], [3]. The second is to develop a network overlay. In the latter, the NV platform directly programs the virtual switch and physical switches to create and coordinate virtual networks. In the former, the virtual switch is replaced with a hypervisor (e.g., KVM) or run on a VM as a terminating point for the virtual network.

Another motivation for network virtualization involves the deployment of new services. One such service includes dynamic workload placement that creates isolated virtual networks for each of the tenants in a multi-tenant data center [53]. A second includes centrally managed security policies that provide dynamic security enforcement for each virtual network [53]. Other motivations for network virtualization include disaster recovery and manageability. Moreover, research in NV is still growing to offer a wider breadth of solutions and beneficial innovations.

In one survey [3], when asked about the greatest benefits of network virtualization (NV), the respondents' top answers included the following:

1) Flexibility
2) OPEX Reduction
3) Scalability
4) CAPEX reduction

Likewise, as indicated in [3], stakeholders of NV most often seek to deploy it for cloud management platform integration, improved performance, better scalability, and richer L2/L3 feature sets. Other work [2] also lists improved disaster recovery times and overcoming limitations of VLAN as key reasons for using SDN in conjunction with NV. In many

cases, networking virtual machines (VMs) are connected together via layer 2 platforms (e.g., Open vSwitch [90]) and configured remotely using JSON or OpenFlow commands [53], [102], [103]. These virtual networks allow for rapid, at scale, innovation that is free of vendor-centric solutions [53].

Within the context of network virtualization, multi-tenant networks transit an abstracted virtual network that sits atop a physical network, sharing multiple racks and services (e.g., controls, acceleration, security, etc.) within a data center [3]. Via direct programming or overlays, the SDN controller helps to set up appropriate virtual networks by monitoring the network topology and configuring the network ports accordingly. The controller also creates the service chains needed to insert the L4-7 services. Consequently, NV also allows researchers to conduct experiments and share resources on production networks while still being isolated from production traffic, which provides added support for enforcing network change management programs.

Beyond sharing resources, virtualization also allows for isolation between VMs, a means to aggregate resources to accommodate greater requirements, the ability to dynamically respond to user mobility with reallocated resources, and simpler management through uniform interfaces [104]. When considered with SDN, possible applications include graphic engineering, security, policy, or network virtualization [2]. Likewise, SDN in NV potentially allows for deep programmability of network infrastructures in order to quickly modify network behavior and provide better policy controls via rich applications [2].

Other areas where NV and SDN are merging include Virtual Edge and SD-WAN, dynamic interconnects, virtual core and aggregation, and data center optimization to name a few [3]. For Virtual Edge and SD-WAN, the controller configures data flows on either the cloud or customer-premise network infrastructure. These actions include configuring appropriate ACLs or QoS settings for subscribers with WAN access to branch networks and services. With dynamic interconnects, the controller sets connections as needed. Virtual core and aggregation utilizes the controller to provide virtualized network segments to support multiple vEPC and maintains network fabric connections across each of the infrastructure services. And, for data center optimization, the controller uses real-time network analytics to maintain the network's topology and modify paths or ensure service level agreement or optimization goals are enforced.

Some existing and open SDN-based NV solutions already exist for network operators. These include FlowVisor [105], FlowN [106], and AutoSlice [107]. Each of these architectures are discussed in [2]. Likewise, a plethora of vendors are also now seeking to compete in the NV market. From one survey [3], we list the lead NV vendors having 6% or more of the market by order or their market share below.

1) VMware NSX (38%)
2) Cisco Systems ACI/Nexus/VTS (21%)
3) Juniper Networks Contrail (21%)
4) Brocade (9%)
5) Nuage Networks (9%)
6) Ciena (6%)
7) Ericson (6%)
8) Huawei (6%)
9) NEC Programmable Flow (6%)

However, there are still many open research questions for NV. For instance, some NV solutions encounter performance issues with control plane over saturation or when the data plane unnecessarily replicates broadcast, unknown unicast, or multicast traffic across the physical fabric [3]. As a result, it still uncertain whether NV solutions can scale to support tens of thousands of servers or not. Additionally, most monitoring and analysis implementations are relatively basic, focusing on data capture and still need more mature capabilities to improve analysis [3]. Furthermore, the monitoring and troubleshooting of large numbers of virtual networks is still a challenge expected to last yet many more years.

Finally, vendors and enterprise participants already have some idea for the directions they hope to take NV/SDN. By order of interest, these include 1) creating virtual networks for cloud applications, 2) WANs for enterprises (SD-WAN), 3) Virtual networks for enterprise campuses, 4) improving flexibility and agility for internal operations, 5) CPE solutions for residential and commercial subscribers (vCPE), and 6) mobile core applications [3]. Another issue resides with network operators increasingly being burdened by the challenge to increasingly accommodate new services and greater bandwidth requirements [108], [109]. For this set of problems, researchers also seek to employ network functions virtualization (NFV), which we will discuss in §VI-B.

### B. NETWORK FUNCTIONS VIRTUALIZATION (NFV)

Network Functions Virtualization (NFV) seeks to utilize virtualization technology to emulate functions that traditionally run on a specialized hardware (or network devices) on high-volume, commodity, servers, switches, and storage devices [110]. By doing so, network operators can replace proprietary middleboxes and network devices with virtual network functions, which can be moved to various network locations and instantiated as required without the need for new equipment installations. Doing so reduces CAPEX by consolidating servers and replacing high-end, purpose-built, devices with commodity hardware to reduce equipment costs. Likewise, OPEX is reduced by minimizing the number of specialized middlebox managers and reducing power requirements [111]. Moreover, OPEX costs can be further reduced through software-based orchestration while still providing network operators with greater opportunities for innovation [109], [112]. Better yet, NFV solves many of the challenges facing network operators as discussed in §I-A.

These network devices include routers, gateways, firewalls, QoS monitors, video transcoders, service level agreement (SLA) monitors, WAN accelerators, etc. [111], [113]. A greater variety of network functions considered for NFV can be found in [2], [114]. Through NFV, these devices can be instantiated, orchestrated (programmed), moved, run, or

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



shutdown on a variety of server platforms, as needed, without additional installation requirements or new equipment. NFV is already highly utilized in data centers, and it has been key to the success of cloud computing solutions [3]. Even now, network users shop, work, learn, socialize and enjoy leisure activities in virtual environments. Likewise, engineers and scientist have virtualized memory, LANs (VLANs), private networks, computers, and much more. Even so, these virtual devices and activities require large amounts of compute, network, and storage. Since it is not feasible to dedicate individual hardware for just one service, NFV allows for virtualized elements to share resources among many devices that might otherwise be underutilized. Hence NFV also supports multi-versioning and multi-tenancy of network functions [2].

By coupling NFV with SDN, network operators can also offer control plane services, like content-centric networking, on-demand virtual networks, and binding of cloud networks via SDN while NFV provides data plane services such as parental control, NAT, WAN acceleration, and web caching [109], [112]. In fact, according to the SDN Market Report produced by SDxCentral [16], this combination of SDN and NFV marks the first significant inflection point in both networking technologies and overall business landscape in the last 20 years.

While NFV predates SDN and does not require it, SDN makes it much easier to orchestrate a multi-tenant data center because it is simpler to implement an SDN switch than a physical switch [53]. SDN also makes it possible to run a separate controller for each virtual network to enhance controllability. As a result, SDN shows great promise for cloud orchestration and networking. It even shows great potential for implementing security and middlebox functionality [115] in new and novel ways within clouds. As it happens, cloud providers are currently the largest consumers of SDN-NFV-based networking technologies, and this is expected to continue through 2020 [16]. Likewise, [16], states that the use cases provided by the confluence of cloud adoption and IT convergence will continue to drive demand for SDN-NFV technologies.

Proponents for NFV also point to SDN as a means of enhancing performance, facilitating operation and maintenance procedures, and simplifying compatibility issues within current deployments [110]. Essentially, SDN allows virtualized devices to be orchestrated as a single system. Network operators currently have to purchase a plethora of middleboxes to perform security, load balancing, and traffic engineering functions, yet NFV offers a distributed compute pool where middlebox functionality can be dynamically installed and orchestrated via SDN. Thus, NFV is able to establish a unifying control framework for deploying middlebox functions throughout the network.

Consider the difficulty of utilizing external network management software within virtual or cloud environments. It may be necessary to accommodate VMs belonging to multiple clients, requiring separate virtual LANs (VLANs), in one physical machine [104]. Similarly, organizations consisting of multiple data centers require a means to move VMs from one site to the next while maintaining IP configurations. Using external network management tools for such feats is no easy task. For instance, migrating multiple virtual machines through traditional methods can take some data centers weeks (or even months) to complete [116].

TABLE 5: Table of SDN-NFV Use Cases adapted from [16]

| Use Case | Sub Use Cases |
|---|---|
| **Network Access Control** | **Campus NAC**<br>**Remote Office / Branch NAC**<br>**M2M NAC**<br>**Unified Communications Optimization** |
| **Network Virtualization** | **Data Center Virtual Networks**<br>**Campus / Branch Virtual Networks**<br>**Data Center Micro Segmentation**<br>**Network Functions as a Service** |
| **Virtual Customer Edge** | **On-premises vCPE**<br>**On-premises vCPE (OTT)**<br>**vCE (Telco)**<br>**vCE (OTT) (a.k.a. Cloud CPE)** |
| **Virtual Core & Aggregation** | **vEPC & vIMS**<br>**vPE**<br>**GiLAN**<br>**Mobile Network Virtualization** |
| **Data Center Optimization** | **Big Data Optimization**<br>**Mice/Elephant Flow Optimization** |

Already, many use cases have been identified for SDN-NFV architectures. They include Network Access Control, Network Virtualization, Virtual Customer Edge, Dynamic Interconnects, Virtual Core and Aggregation, Data Optimization, and other unspecified contributions [16]. For which, revenue values are expected to triple from $18B in 2015 to approximately $65B in 2018 (next year) [16]. The sub use cases for the above use cases are given in Table 5; however, the reader is referred to [117] for use case definitions.

Another enabler for NFV is containerization, which allows for container isolation to prevent multiple tenants from influencing each other's applications within an operating system. Containers (e.g., Linux containers) also offer an alternative to various hypervisor approaches, since they utilize less overhead by interacting with the same Linux kernel as the host system to achieve greater performance. Additionally, by using container runtime image management solutions, like Docker and Exo-Clone [118], [119], these containers can access directories that are not part of the container to exchange data between containers or store data beyond the container's shutdown.

Still, multiple issues for container based NFVs still exist. For instance, even when memory limitations are imposed on containers, the Linux *free* command might indicate that memory exists within the container, when it actually refers to the entire host system [120]. Similarly, attempting to retrieve CPU utilization and availability generally gives notice for all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



CPUs available to the host, regardless of *cpuset* applied to the container [120]. Live migration is yet another problem of containers. This prevents network operators from moving a running container to a different host system, and it poses an interesting challenge for future research. Currently, it is more convenient to simply restart a container on different host machines, which is still faster than restarting a virtual machine [120]. Restarting containers also requires the updating of the network infrastructures routes to ensure applications are correctly routed to the correct container. This also poses interesting opportunities for NFV-SDN research. Further, containers also suffer from potential *fork bombs*, where a process forks via a fork system call to create a child process with all the memory of its parent process [120]. The result is a potential DoS of the host kernel. Finally, unikernels and rump kernels can also re-use the existing management tools of virtual machines and serves an interesting subject for future work [120].

Clearly, NFV is not without its challenges and additional research is still needed to enable network operators to better deploy SDN-NFV systems. Research topics include portability and interoperability, integration, performance, legacy platform compatibility, management and orchestration, automation, stability, and security and resiliency [110]. In fact, early NFV deployments were beset with performance issues [16]. Now, with most virtual infrastructures emphasizing isolation boundaries between virtual machines, they still fail to provide the advanced resource management needed to fairly allocate compute, I/O, and storage resources [16]. Consequently, co-located VMs (acting as a "noisy neighbor") can consume too much of the available resources. The result of this issue is that network operators cannot comfortably migrate from their predictable proprietary hardware to NFV solutions [16].

Mijumbi et al. [24] also recently identified six open challenges and research opportunities for advancing SDN and NFV. While we provide a brief description below, we refer the reader to [24] for further details.

**Resource Management** Since servers, including finite amounts of memory, compute, and storage capacity, may be distributed across multiple domains, inter-domain link capacity is also finite. Hence, dynamism, scalability, and automation must be applied to management of these resources to achieve economies of scale. Three challenges are specifically identified within this context. Those are a) NFV PoP locations, b) function placement, and c) dynamic resource management.

**Distributed Management** Current management and operation (MANO) approaches, defined by ETSI, focus on centralized solutions. The result is scalability limitations, which is especially troublesome for multi-domain services due to communication overhead and process delays. As a result, research opportunities include the development of efficient monitoring mechanisms that better react to dynamic demands and changing service requirements, while also providing information required for dynamic change configurations to distributed entities. Likewise, lightweight communication protocols for optimizing resource usage and service performance represent great research opportunities.

**Management of SDN** While SDN and NFV are highly complementary, individually, their dynamism and variability serve to curtail the human operators' visibility and control. Consequently, management approaches must be developed beyond simply managing the virtualized compute resources and functions. Instead, management solutions that combine SDN and NFV serve as key research areas. However, we believe that OpenBox [121] and Slick [122] represent great strides in this arena. Other research opportunities in this field include the management, location, number, and conflict resolution in cases where the control plane is physically distributed.

**Management across the board** Another challenge of NFV is its support for fault, configuration, accounting, performance, security (FCAPS) functions [24]. For instance, accounting management for tracking network utilization and billing is still completely overlooked in nearly all systems, while security and performance management appear in a limited context [24]. Generally speaking, an entire service life-cycle management system is still missing. And, this lack of management runs contradictory to the realization of multi-vendor, coexistence on open NFV platforms. It also misses out on one of NFV's unique selling points of automation for the set up and tear down of service chains, such as billing, configuration, optimization, performance, and response to faults. Thus, research providing support for FCAPS management is still needed to enhance the use of NFV in a telco.

**Programmability and intelligence** For NFV and SDN, intelligent and automatic mechanisms for transforming high-level policy are needed to apply operational parameters and validate the integrity of configurations. The goal here is to allow NFV to be successfully deployed and maintained over heterogeneous physical resources while still providing complex resources. Ultimately, this will require the development of a rich set of programmable interfaces to extend SDN functionality beyond the simple control of switch devices. Again, we point to OpenBox [121] and Slick [122] as recent developments in this field, but the interfaces for these frameworks has yet to reach the level of programmability as described above. Likewise, other abstractions are needed so network functions can be instantiated across multiple vendor technologies and allow for their dynamic (re-)programming and placement. Finally, another research challenge hinges on enabling intelligence in MANO NFV systems that are capable of (re-)configuring operations to react to events at runtime.

**Interfacing and interoperability** If NFV is to break the bond between telecommunications service provider (TSPs) and equipment vendors, then one key requirement it must achieve is interoperability across different vendors and different functions. In work surveyed by [24], all NFV MANO related projects observed some level of interoperability problems, with each using a custom model and/or representa-

tion for services and functions. This observation means that chaining functions from different operators into a single service is impossible without clearly defined interfaces, which is primarily because ETSI does not yet describe a data model to realize descriptors. As for those that do exist (e.g., the Alliance for Telecommunications Industry Solutions (ATIS) [123]), they only consider generic descriptors and lack any technical requirements for enabling use cases.

These challenges will likely drive the technical focus of SDN-NFV frameworks in future research. Additionally, putting these systems in the hands of network operators to develop additional use cases for SDN-NFV systems is paramount to their adoption. One might imagine a fully portable network capable of being rapidly deployed worldwide with minimum network equipment as a use case. Furthermore, while on-demand scaling and provisioning can be improved for NFV, other problems–like centralized management for each NF–still exist.

### C. SOFTWARE-DEFINED NETWORK FUNCTIONS VIRTUALIZATION (SDNFV)

One of the issues with traditional middlebox and NFV devices (when utilized as monolithic middleboxes) is that they still suffer from management and multi-tenancy issues. In both cases, they are treated as monolithic devices having proprietary software already installed, which serves as a hindrance to on demand scaling and provisioning. Granted that cost of ownership and management and on-demand scaling and provisioning are often improved by using NFV over traditional middleboxes, NFV alone still does not provide centralized management. As a result, network traffic often traverses a service chain (or a sequence of NFs) to achieve desired results. This chaining, where multiple NFs share similar processing steps, contributes to greater delays and decreases throughput [121].

In answer to this problem, OpenBox [121] recently proposed a framework and a protocol that makes network functions software-defined, having their own logically centralized controller. Whereas both hardware and virtual network function solutions were each treated as an individual piece of hardware, the OpenBox solution allows for network functions to be combined as a single entity, which alleviates some of the redundant processing steps that occur with service chaining. The OpenBox [121] approach to merging network functions demonstrates throughput performance improvement of up to 90% with latency improvements of 35%-50%.

Another work having nice overlaps with OpenBox [121] is Slick (SDN + Click) [122]. While Slick focuses on placement of network functions and steering traffic to them, OpenBox focuses on combining network functions in a way that avoids repeated filtering. Slick also appears to be farther along in its implementation than OpenBox. Still, Slick's approach might offer greater performance if it incorporated OpenBox's approach to combining functions–especially in locations where Slick has co-located elements. Additionally, OpenBox's future work seems to indicate that they hope to eventually address smart allocation (placement) and traffic engineering (steering), which is something the creators of Slick already address.

However, while the OpenBox protocol offers 40 types of abstract processing blocks (i.e., network function applications) and Slick offers 15 distinct network functions, introducing new processing blocks is not a simple process. In both cases, one must first write a Click [124], [125] module in C++ and then implement a translation module in Python. Hence, these frameworks could benefit from a high-level programming language offering comparable abstractions to what Pyretic [40] provides for POX [39] and Ryuretic [42] provides for RYU [41].

### D. CLOUD

Of the virtualization technologies, the cloud is possibly the most abstracted of the group as it comprises aspects of NV and NFV, and more recently, SDN. Cloud environments seek to leverage best practices for leveraging software-based solutions, micro-services, virtualized commodity platforms, elastic scaling, scalable services, and service composition to allow more rapid innovation for network operators [109]. Likewise, many of today's cloud environments are typically orchestrated by systems like OpenStack, CloudStack, vRealize and Microsoft SCVMM [3]. Of course, rapid development and refactoring is still occurring with many orchestration stacks, resulting in less stability with networking APIs. As a result, alternate means for integration management are still being considered.

Already, the trend of including SDN controllers and switches with cloud orchestration tools has seen rapid growth [2]. One such solution, met for telco companies, is CORD [109]. In this work, researchers observe that at least one major telco has 4700 central office (CO) locations, with each containing up to 300 unique network devices. As a result, these devices represent a significant barrier to innovation both in terms of capital expenditure (CAPEX) and operational expenditure (OPEX). Consequentially, Peterson et al. offer CORD [109] as a telco architecture that combines SDN, NFV, and elastic cloud services to build cost-effective and more agile access networks.

Never the less, an open challenge and frequent concern for cloud environments is security. The programmability that SDN and NFV bring to cloud architectures also invites new attack vectors with potentially farther reaching impact than in non-virtualized environments [24]. According to the ETSI NFV report [100], clouds face generic virtualization threats (e.g., interrupt isolation, memory leakage, etc.), physical system and network function threats (i.e., flooding attacks, routing security, etc.), and new threats created by the combination of virtualization technologies with networking. Some of these threats have recently been outlined by the ETSI NFV security group, and we list these key issues below.

1) Availability of Management Support Infrastructure
2) Secured Boot

3) Secure crash
4) Isolation of Multiple Administrators
5) Performance isolation
6) User/Tenant Authorization, Authentication, & Accounting
7) Authenticated Time Service
8) Cloned Images Containing Private Keys
9) Back-Doors via Virtualized Test & Monitoring Functions
10) Topology Validation & Enforcement

However, as Mijumbi et al. [24] observe, the ETSI NFV report [100] does not offer recommendations for addressing them. Currently, only CloudBand [126] offers a security solution having anomaly prediction, detection, and isolation, while also offering security as a service. Other cloud architectures, highlighted by [24], such as Zoom [127] and Planet Orchestrate [128], merely claim security support based on best practices and product integration [24]. Hence, real security support is insufficient in all NFV products, in spite of legitimate new threats. Thus, these topics represent great opportunities for cloud research to further advance SDN and NFV use cases for network operators. Likewise, detecting and blocking possible intrusion in cloud environments and maintaining isolation to protect one TSP's data and configuration information from other TSPs in multi-vendor environments represent important security challenges.

## VII. SECURITY

Like its predecessor, work in SDN has moved forward with little regard to security. As a result, this area of research presents vast opportunities for researchers. It also leaves much to be desired by network operators. SDN security essentially falls into two categories. First is the security of the SDN architecture itself. One can imagine a "stuxnet-like" [129] virus designed specifically to hijack, shutdown, or corrupt SDN controllers, and the impact such a virus could have on a network. Second is security within networks–detecting and preventing malicious attacks on end users. Some researchers have argued that commercial adoption of SDN is still deterred by its ability to offer security and dependability [130]. However, SDN is not alone with its security challenges. Traditional networks, have their own issues, and recent reports indicate that the control plane of traditional networks is no longer as safe as once thought [131].

### A. CONTROLLER SECURITY

By separating the control plane from the data plane, new security challenges arise for SDN architectures. Already numerous attack vectors have already been identified [14], [130]. These include issues with the SDN controller's APIs, memory, and various other factors as we will now discuss.

For instance, several security issues exist with OpenFlow-based networks [132]–[134], which are susceptible to a variety of security and dependability problems that include tampering, repudiation, information disclosure [132], spoofing [132], privilege escalation [132], and denial of service [14], [132], [135]. Since many controllers utilize OpenFlow, they too have comparable security and resiliency issues. For instance, controllers such as Beacon [43], Floodlight [44], Maestro [62], OpenDayLight [1], and POX [39] are all susceptible to fake topology, spoofing, tampering, repudiation, information disclosure, DoS, TCAM exhaustion, and privilege escalation [15], [28].

Alas, while it is still unclear what software might take advantage of this, another issue with the SDN controller's northbound interface is that most APIs frequently leave the controller exposed to other applications or management software [2]. Another issue is that a simple value change in memory can drastically affect the reliability and operation of these controllers [15], [133]. Reliability is further discussed in §VIII-B, yet these issues remain open challenges in SDNs. Other issues with the SDN control plane are identified in a survey by Kreutz et al. [15], which includes the following six vectors, three of which are unique to SDN architectures. Those threats include 1) DDoS attacks, 2) attack inflation, 3) exploitation of logically centralized controllers, 4) compromised controllers which compromise the entire network, 5) malicious controller applications, and 6) negative impacts on recovery speeds and fault diagnosis [15]. Kreutz et al. [15] also point to weaknesses in access control, isolation, protection, and security as contributing to the above-mentioned SDN threats.

Some countermeasures include access control, flow aggregation, attack detection, event filtering, firewall and IDPS, forensics support, packet dropping, shorter timeouts, and rate limiting–most of which are not yet supported by SDN architectures [15]. Other work [28] has proposed the incorporation of a shim layer (or virtual middlebox) between the control and data plane to protect controllers from such attacks. Security extensions for SDN controllers have been proposed as well [136]. More research for controller security, protection, and isolation mechanisms is needed to address network operator security concerns with SDN. Such mechanisms must also be sandboxed from other applications and from the controller. Furthermore, methods to establish trust between controllers are needed, both to ensure proper forwarding and to detect malicious elements before a misconfiguration can occur and damage the network [15], [28], [136]. Malware scanners for SDN applications are one possibility, so too are certificate based authentication, authorization, and accounting systems [137]. Both are relevant since few controllers even utilize secure TCP connections.

To further help mitigate problems with malicious controllers, Schiff and Schmid [78] analyzed distributed control planes that are resilient to malicious controllers, which are represented as a malicious network administrator, a compromised controller software, or an unintentional misconfiguration. The authors argue that a control plane that is resilient to malicious controllers requires a basic notion of memory and awareness of history. Thus, they introduce a model in which most benign controllers are responsible for accurately updat-

ing data plane switches despite the presence of malicious controllers by using a light-weight in-band communication mechanism to achieve consensus. Their model, however, assumes that data plane switches are trusted and each switch maintains a summary of the controller state and history. As we will soon discuss, this may not be a valid assumption. After verifying that a majority of controllers agree on the change, the switches implement the requested update.

To alleviate trust issues between the SDN controller and its applications, Betge-Brezetz et al. [138] rely on several redundant controllers that may also be running in separate executing environments. However, instead of a consensus protocol, the authors introduced an intermediary layer (a trusted-oriented controller proxy (ToCP)) between the control plane and the data plane. The ToCP is responsible for collecting and analyzing configuration requests from all redundant controllers and evaluating if they are consistent and trustworthy before allowing their placement in the data plane. However, their results show that ToCP imposes a performance cost due to service degradation and additional computing requirements.

Compromised switches are yet another issue beyond that of the SDN controller. One architecture that has addressed this issue is FlowMon [139]. This architecture detects compromised switches through real-time analysis of network traffic statistics collected by OpenFlow in an SDN controller. Their main objective is to detect packet droppers (i.e., switches that purposely drop packets) and packet swappers (i.e., switches that forward packets to a port for which they are not intended). To do so, the authors add two additional functional blocks to an SDN controller. One being a malicious switch detection and prevention (MSDP) block, and the other a policy block. While the MSDP continually and transparently analyzes communication between the controller and switches, the policy block contains a set of rules that are triggered anytime a malicious switch is detected. The authors also propose algorithms for detecting packet droppers, using information collected from port statistics from switches, and packet swappers, by investigating the reports of unknown flows and comparing their expected output interfaces to their observed ones. Results indicate that both algorithms are able to detect malicious switches in a mixed environment.

### B. TRADITIONAL NETWORK SECURITY

Meanwhile, cyber-attacks are of great concern across all networks as they push the boundaries of both traditional networks and SDNs. New applications are needed to detect and prevent these attacks. Such threats include advance persistent threats (APTs), data exfiltration, malware propagation, denial of service, and many others. In response, organizations, such as the Open Network Foundation (ONF), have taken an active role in addressing this issue by founding the ONF security working group. Equally important is determining which level (or tier) of an SDN to deploy these security applications in order to avoid performance degradation and unmanageable false-positive alarms.

Our own research focuses on leveraging the capabilities of software-defined networking (SDN) to exploit the state and header information available to edge devices on government, industry, and campus networks in order to mitigate or eliminate existing attack vectors [42], [77], [140]–[142]. Network Flow Guard DHCP (NFGD) introduces an extensible module for detecting and preventing Rogue DHCP servers [140] and Network Flow Guard ARP (NFGA) introduces a similar module for detecting and preventing ARP poisoning [141]. The latest security solution introduces a Trusted Agent to assist the SDN controller with identifying rogue access points (RAPs) via passive and active detection measures [142]. While NFG offers an excellent addition to network security and potentially reduces the number of middleboxes required on a network, it is still only intended as an initial barrier or first defense in a defense-in-depth strategy. As such, it has little application for security within the network infrastructure. Other SDN-based security features are discussed below.

#### 1) Network Access Control

Another aspect of network security tailors to network access control (NAC), which is used to set appropriate privileges for users or devices accessing the network. As a result, access control limits can be applied to clients and appropriate service chains and quality of service guarantees can be applied for each client. The SDN controller facilitates these actions by configuring appropriate match:action rules for underlying switches based on access control lists and quality of service agreements. In some cases (see Sec. V), the controller even sets up the service chains to support L4-7 services.

After observing that many ACLs were either static or require repeated network operator involvement to update, Cox et al. [77] developed an SDN solution to automate the revocation of security policy enforcements through the use of a Trusted Agent. Their method allows an automated system, the Trusted Agent, to modify the SDN controller ACL in lieu of the network operator. This work has also been expanded to include active testing measures to enhance security on local networks.

#### 2) Cryptographic Devices

According to top military officials, one arena where SDN will receive government support is with cryptographic devices. The Army's Chief Information Officer recently released a report calling for the Army to drive research and development efforts towards extending SDN to such devices [22] in order to support regional and redundant control options.

#### 3) Random Host Mutation

The use of SDN to provide random host mutation or a transparent moving target defense (MTD) recently gained significant attention at the 2016 Military Communications conference. MTD transparently mutates IP addresses in ways that are both highly unpredictable and frequently changing while still maintaining configuration integrity and minimal operational overhead [143]. Doing so serves to minimize the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



time that a host computer spends at one location (i.e., IP address) being exposed to attackers. Other research [144] has also already determined that there exists an optimal IP address change rate for thwarting attacks as well as an optimal attack crafting time.

While the goal of this work is to thwart scanning using a random and unpredictable mutation of host IP addresses, other applications of this work could potentially thwart DDoS attacks. Still, other research is needed to fully leverage this work in tactical networks. For instance, in military networks and other resource constrained networks, the controller communication will need to occur in-band.

#### 4) Security Applications

In this paper, we have hardly touched upon the sheer volume of SDN security work that has already been produced. Yet, we expect the number of applications offering security solutions to grow significantly as SDN obtains greater traction in production networks. Additionally, this will also offer opportunities for researchers to transition their solutions to marketable offerings through app stores (e.g., HP [145]). As a result, methods for vetting and transitioning security solutions from research to commercially viable network applications are needed. Likewise, ensuring such apps are able to work together, share state, and not crash the SDN stack are all worthy research efforts within this field. Hence, developing tools for testing applications and ensuring their reliability and fault-tolerance will further serve to advance SDN's adoption.

## VIII. TOOLS, FAULT-TOLERANCE AND OTHER SDN ANALYSIS CONSIDERATIONS

While a variety of tools are emerging to assist network operators and researchers with evaluating SDNs, concerns over reliability and fault tolerance continue to hinder SDN's adoption [72]. In the following sections, tools are given a cursory overview, while greater attention is given to fault tolerance, performance, and scalability.

### A. TOOLS

As tools are covered in other literature, and in much greater depth, we provide a cursory overview of these tools via Table 6. We offer this list to point researchers to tools that are available to enhance their own research, promoting the advancement of SDN. Undoubtedly, this list is far from complete as new tools are frequently being developed. However, we feel that listing these tools by classification and referring the reader to the resources for better understanding them provides for a more complete survey. Hence, these tools are available to assist network operators and researchers with debugging, verifying, and testing network flows, however, since other work [2], [146] has already discussed these tools in greater depth, we refer readers to these resources for a better understanding of their application.

TABLE 6: Tool list by common headings [2], [146]

| Common Headings | Tools |
|---|---|
| **Performance Testing** | OFLOPS, Cbench, iperf, MoonGen |
| **Data Plane Verification** | Anteater, Header SpaceAnalysis (HSA), FlowChecker, VeriFlow, NetPlumber, NetSigtht, ndb, OFTest |
| **Control Plane Verification** | NICE Framework, FlowLog, Cbench |
| **Network Debugging** | ndb, HSA, OFRewind framework, Pip, NICE-OF, OFTRACE |
| **Protocol Verification** | Proof assistant Coq tool |
| **Property Verification** | Anteater, HSA, FlowChecker, VeriFlow |
| **Reachability Analysis and Loop Detection** | HSA, VeriFlow, NetPlumber, APVerifier |
| **Isolation Verification** | AP Verifier and Splendid Isolation |
| **Configuration Management** | FlowChecker, AntEater, ConfigChecker |
| **Network Security** | FLOVER |
| **Automatic Synthesis** | Reactive Synthesis and Model Checking |

### B. FAULT TOLERANCE

In research conducted in [72], the authors observe that the complexities within SDN applications (apps), coupled with buggy switches, lead to a number of bugs, which include timing bugs and null pointers. Moreover, given the recent success of open-source controllers, like OpenDayLight [1], ONOS [33], Ryu [41], etc., and the emergence of SDN App stores, like HP's SDN App Store [145], it is expected that greater numbers of apps will be offered with only limited testing by third parties. However, SDN suffers from fate-sharing relationships between SDN apps and their controllers or SDN apps and the network wherein failure in the first example leads to a mutual crash of the controller and its applications, while the second leads to network safety violations (e.g., network-loops, black-hole, etc.) [72]. Further analysis in [72] shows that bugs even exist in the SDN apps that come bundled with controllers, like OpenDayLight, by default. As a result, it is no surprise that some network operators are hesitant to adopt SDN.

In order to improve the fault tolerance of SDNs, [72] observe that three directions are primarily followed. The first involves diagnosing and pinpointing the failures root cause. The second involves providing better programming abstractions for developers to avoid the root cause of failures (see also §III-B), and the third involves improving fault-recovery techniques through controller replication. Of these, only the third is identified as a means to recover from SDN app failures in production networks, since generated replicas can transparently take over the control of the network in the advent of a primary controller failure. However, the authors

of [72], also point out that controller replication does little for SDN app crashes caused by deterministic bugs.

The authors of [72] also discuss three challenges to recovering SDN apps from failures. First, network state is modified and shared amongst multiple SDN apps, which creates a challenge for maintaining consistent state amongst all apps during a failure recovery. Second, there are no protocols allowing network operators to exploit the semantics of SDN control messages to design better recovery mechanisms, which they identify as a hard problem. Third, controllers, being monolithic in their design, can often fail even if only one of its components fails. As a result, component isolation is also required.

Having identified the above challenges, LegoSDN [72] is offered as a fault-tolerant controller architecture. By sandboxing SDN apps, the failure of one app is constrained to the sandbox where it runs, and interactions are communicated via remote procedure calls (RPC). The authors also utilize a FloodLight controller to support LegoSDN and allow network operators to use a familiar API.

Similar to LegoSDN is Ravana [147], which ensures transactions are executed only once across replicas, which allows it to correctly handle switch state, while also avoiding rollbacks or repeated command executions. However, Ravana requires additional OpenFlow extensions for its participating switches to guarantee correctness. Other related works focus on recovery after a controller failure–typically by applying Paxos [72], [148]. Beyond handling controller failures, other approaches like Pyretic [40] and Ryuretic [42] seek to provide more efficient programming frameworks for creating SDN apps, which can potentially minimize or eliminate bugs. Yet, these frameworks do not address isolation of SDN apps, nor do they offer checkpoint and recovery support for SDN apps. Hence, applying LegoSDN's measures for avoiding redundant or conflicting rules to such frameworks serves as a future research direction.

Currently, many opportunities exist for advancing fault tolerance in SDNs. While not a complete list, the following bullets represent some areas where researchers can make immediate impacts.

- Novel abstractions for minimizing bugs. While bugs exist in most applications, bugs in an SDN app can crash the entire SDN stack [72].
- Algorithms and frameworks for maintaining consistency across both control and data planes to ensure safe recovery [72].
- Isolation of SDN apps–allowing them to run in spite of a controller failure, even if unable to interact with the network [72].
- Minimize SDN application developer efforts to avoid code changes when implementing fault-tolerant architectures [72].
- Fast or timely recovery to avoid violations with service level agreements (SLAs) [72].
- Creating methods or algorithms that are applied to both network functions virtualization and SDN controllers.
- Frameworks addressing race conditions, atomicity violations, deadlock, livelock, etc. [149].
- Algorithms to troubleshoot bugs and carefully schedule replay events after an SDN app failure occurs [149].
- More efficient rollback algorithms to maintain consistency across different SDN apps [72].

### C. SCALABILITY AND PERFORMANCE

As with other networks, network operators must consider scalability and performance in their SDN deployments. While scalability challenges are not unique to SDNs (even traditional networks struggle with convergence and consistency requirements) they are a point of contention [150]. As we will next discuss, scalability and performance concerns for SDNs lie with both controllers and their switches.

Controller scalability problems primarily arise from three issues [14]. First is the latency that develops as a single controller reacts with many nodes [14]. Second is communication methods between peer, supporting, or slave controllers via east-west bound APIs [14]. Third is the size and operation of the controller's backend database. Research in this arena will undoubtedly focus on limiting communications between network nodes and the controller and reducing the size of the controller's database [14]. Likewise, exploitation of parallelism in multicores to improve I/O performance may be adapted [150]. Yet, another solution may include the ability to physically distribute the control plane elements and still maintain a network-wide view, as is done in Onix [63] [150].

The number of rules that can be installed on an OpenFlow switch also raises scalability concerns. Some work [87] seeks to optimize how limited rule-table space in hardware is used. Some recommendations include adding a network processor in the data plane of hardware switches, utilizing a software agent on the hardware switch, or incorporating software switches [87]. But, it is still not known whether these compromises or some other course of action will resolve the hardware scalability issues of OpenFlow switches.

Similarly, performance improvements require faster software and hardware forwarding techniques. Improved filtering mechanisms are needed as well to ensure better efficiency and scalability of controller processing cycles in order to avoid the repeated computation of policies that also affect performance [70]. Partitioning of larger networks into smaller subnetworks may also be called for to improve performance and scalability issues [76], [151]. Consequently, network partitioning and distribution of controllers will drive research towards vendor-neutral and controller-neutral east-west bound interfaces [76].

As an aid to evaluating SDN performance, network operators can also turn to various benchmarks. Most fields of engineering incorporate gold standards or benchmarks to validate their proposed improvements. Computer architecture, wireless networks, fault tolerance models, traditional networks, and many others all utilize established benchmarks to compare new architectures, protocols, and methods to existing capabilities. SDN is no different, and there have developed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



several benchmark systems from very simple (Cbench [152]), to academic (OFCProbe [153], and commercial (Spirent OpenFlow Suite for Test Centers or IxiaâĂŹs IxNetwork) for SDN architectures.

Still, testbeds are also needed to validate the scalability, performance, and fault tolerance of various solutions, and to establish use cases for change management purposes. Unfortunately, hardware is not cheap and network failures are not acceptable, which makes testing on a live network more difficult to accomplish. Thus, simulation and/or emulation platforms are needed to provide researchers and engineers with a means to validate their solutions before applying them on a live network.

## IX. MODELING AND SIMULATION

Numerous tools are available within the realm of SDN that permit both troubleshooting and performance analysis of both live and simulated networks. Such tools are also highly valuable for visualizing and evaluating changes to network infrastructure or protocols. They are also valuable for establishing justification in change management programs. However, although briefly discussed in §II and §XII-D, the ease and intuition with which network operators will utilize such tools can be improved. Likewise, the ability of such tools to replicate live network traffic with flexible options can still be expanded.

For instance, virtualization provides the most immediate yet tedious mechanism for producing representative network testbeds. Simply instantiating multiple virtual machines (VMs) and then connecting them within a single computer system is a readily available process. Additionally, the VMs are effectively clean slates, allowing an operator to install whichever operating system or software he/she may choose. However, this option is certainly the most tedious and resource-intensive, and as such, is not typically viewed as a viable solution. More automation, visualization, and scalability is preferred.

The predominant tool for creating and testing proposed concepts and topologies in SDN is Mininet [154]. Considered a tool for building "a network in a laptop", Mininet is designed around simple shell processes, which are given their own network namespace and connected within a Linux-based environment by virtual Ethernet (*veth*) pairs. This architecture provides a lightweight option for rapidly prototyping proposed SDNs. The shell processes, which act as the hosts, middleboxes, and controllers of the network, require significantly fewer resources than complete VMs, but only permit the viewing of networks composed of homogeneous systems. Constructing these emulated networks is generally administered via a Python API, which permits the creation and configuration of its various components. Additionally, via the Python Standard Library, specific tasks for network monitoring or traffic generation may be scheduled to execute on the generated processes. As of Mininet version 2.1.0, the MiniEdit [155] GUI has been included which permits the creation and visualization of these configured network topologies.

One of the more intrusive drawbacks to using Mininet is its performance reliability at scale. For an emulated topology of hundreds of components, performance may not necessarily be hindered if memory usage of the underlying system is not fully utilized. However, for larger scales in the thousands of hosts, middleboxes, etc., performance has been shown to degrade as more resources are required to fully realize the underlying components of each node in the topology. This degradation has been demonstrated in prior works [156], [157]. Remedies to this issue are available by limiting the percentage of process space that individual hosts can occupy. With greater numbers of nodes though, the realism of the virtualized hosts can become constrained. Furthermore, limitations are not imposed on the virtualized middleboxes and controllers, allowing them to still use as many resources as they would typically need. Another performance fidelity issue exists in the default connections employed by Mininet which do not provide specific bandwidth limits or quality of service between the *veth* pairs. If more specific link constraints are required, TCLinks executing the Linux traffic control (TC) program may be used. Even so, the main Mininet process is still obligated to operate under the Linux scheduler of the system on which it is run.

Multiple experimental extensions to Mininet have been proposed to address some of its limitations. In terms of scalability, distributed environments have been developed to allow Mininet to operate across multiple computer systems. Referred to as Mininet CE [158], this design permits Mininet networks to execute collectively on different machines as a single emulation. Another limitation of Mininet is its inability to model wireless connections. OpenNet [159] addresses wireless requirements by utilizing Mininet for the wired infrastructure connected to an ns-3 [160] component for wireless/mobility modeling. Connections between Mininet and ns-3 are realized using TAP interfaces. In this way, packets sent from a simulated node in ns-3 may be sent out of the simulation to a real-world recipient, and conversely, packets may be tunneled in through the TAP interface and received within the simulation. OpenNet requires that topologies be coded in both Mininet and ns-3 frameworks. Together, these frameworks provide a high degree of extensibility, allowing for the development and deployment of new experimental protocols. More recently, Mininet-WiFi [161] has been introduced as an alternative Mininet-based emulator of OpenFlow/SDN scenarios, which replicates real networking environments for high-fidelity experiments. Its advantage over OpenNet is that it does not require an ns-3 component, so programming is primarily Python-based.

In addition to Mininet, numerous alternative emulation and simulation tools exist for developing and researching SDN. One commercial option, EstiNet [162] offers both a GUI interface and the ability to model wireless networks. Based on the network simulator, NCTUns, out of the National Chiao Tung University, it provides a unique kernel reentering simulation methodology that allows real applications to run

on nodes in its simulated network without modification. Kernel reentering allows simulated packets to enter and exit the simulation through network tunnels. This design allows the simulator to use the actual network layers of the Linux kernel to process packets. Through this mechanism, it can support the simulation of any SDN controller library with complete portability. As an example, previous work has demonstrated its effectiveness in permitting NOX, POX, and Floodlight controller applications. Other work has also aimed to demonstrate its ability to maintain adequate performance fidelity in comparison to Mininet [163]. However, as proprietary software, it is difficult to adapt to new network protocols as its source code cannot be modified directly. It also limits simulation times to 1000 seconds and nodes to 50. Other simulation and emulation solutions also include CloudSim, NetKit/AutoNetKit, VL2, CORE, and Air-in-a-Box as detailed in [2].

Some other, more experimental ventures in existing simulation tools are also available; however, matriculation cycles at research universities leave some of these tools with uncertain maintenance outlooks. The flow-based simulator *fs* [164] has been extended to support some SDN capabilities in a framework known as *fs-sdn* [165]. This framework is capable of directly incorporating the POX OpenFlow controller libraries and API without modification. POX is the only controller that is compatible with this framework though, and it is limited to OpenFlow 1.0 switch support. The Distributed OpenFlow Testbed (DOT) [166] exhibits a distributed capability similar to Mininet CE, emulating a network across multiple computer systems. Additionally, DOT presents a similar link design to Mininet, employing *veth* pairs. It differs from Mininet CE in design though, implementing a master/slave architecture for node management. The SDN Troubleshooting System (STS) [149] provides a GUI and debugging support to permit troubleshooting of SDN topologies and applications utilizing OpenFlow 1.0.

Various attempts, including the previously mentioned Mininet/ns-3 hybrid OpenNet, have been implemented in the network simulator ns-3 to contribute SDN-based functionality. Within its baseline is an OpenFlow-based device model that provides 0.8.9 support and a non-portable controller interface [167]. More recently, an OpenFlow 1.3 capability has been introduced outside of the mainline ns-3 which follows a similar design to the baseline implementation. This extension provides a significant update to a more practical OpenFlow specification but maintains the non-portable controller design [168], [169]. Another work on ns-3 and its Direct Code Execution (DCE) module [170] provides a framework for directly executing Python-based controller libraries, such as POX and Ryu, from within the simulator. DCE provides a dynamic redirection mechanism that allows network applications to directly execute within the nodes of a simulated ns-3 topology. In conjunction with the OpenFlow 1.3 work, the resulting DCE framework enables portability of Ryu applications supporting OpenFlow 1.3 while still providing the variety of other capabilities available in ns-3 (wireless and LTE support, BRITE integration, real-world network connectivity and emulation, etc.).

As mentioned briefly in this section, VMs on a single system or even a small-scale deployment prove to be a tedious and resource-intensive option for employing a virtual testbed for SDN research. However, ample scale, resources, and infrastructure proves to be a more viable option in national and international collaborative research efforts. This concept is not new, with research environments such as EmuLab [171] and PlanetLab [172] in operation for over a decade. These frameworks provided virtual resources for innovative networking research many years before the initial breakthroughs realized in SDN. More recently, organizations and collective collaborations have contributed testbeds with the specific intent to further the state of the art in SDN. Examples of these endeavors include the Global Environment for Network Innovations (GENI) [173] in the United States, the OpenFlow at Trans-Eurasia Information Network (OF@TEIN) [174] connecting 20 countries in Asia and 34 in Europe, OpenFlow in Europe Linking Infrastructure and Applications (OFELIA) [175], Research Infrastructure for large-Scale network Experiments (RISE) [176] in Japan, the UniCloud project [177] out of Taiwan, and Smart Applications on Virtual Infrastructure (SAVI) [178] in Canada in addition to multiple others. A substantial investigation of the technologies behind many of these infrastructures has been previously conducted in [179]. That work includes design objectives, architecture, and some examples of research conducted in these testbeds while also promoting similar endeavors implemented as the China Environment for Network Innovations (CENI).

Universities and other research institutions have collaborated to provide these environments of virtualized resources that can be provisioned and leveraged for use in research in SDN and other cutting-edge networking paradigms. Participating campuses contribute functional components that act in much the same way as traditional servers in data centers and the "cloud" by allocating resources such as virtual machines (VMs) in a fair and reliable manner. Using SDN, numerous distinct networks can be realized through virtual network slicing. The resulting slices are isolated from one another, guaranteeing the integrity of each created network in terms of its functional characteristics, such as its link parameters, while preventing interference across slices. Furthermore, a diverse array of resource options are available in terms of VM configuration and link specification through a number of tunneling techniques. Deployment of experiments in these testbeds generally involves the following steps: resource specification, virtual allocation, experimentation, and resource release. Specifying the resources that are required for a particular experiment is handled through specification files describing the type and number of resources to be requested and how they are linked together while also permitting more fine-tuned installation behavior. Software installation, service initialization, and other scripted behaviors can be defined for particular compute resources while bandwidth, delay, and other link characteristics can be configured for the network

connections. With a specification created, a user can submit it, and if sufficient resources are available, they will be allocated. As these resources are real as opposed to simulated, they require some time to boot and perform any requested installations or services. Once available though, the experimenter can perform any required experiments, analyzing various SDN controller or switch configurations and generating traffic. Resources are requested for a certain amount of time but may be renewed as demand permits. Upon completion of experimentation, the resources can be returned for use by other experimenters. These types of infrastructure provide the necessary scale that cannot be realized in small-scale emulators as well as more adequate levels of accuracy and reliability that are not achieved in network simulators.

While models and standards, have yet to be established for SDN research, they are needed to demonstrate to network operators how applications will interact within their networks, and how security features will impact their networks [32]. These models could evaluate the exchange of secure information or expose insecure interfaces for applications and other elements. Likewise, such modeling can help address storage, compute, and networking concerns. For example, performance is often considered in terms of processing speed, which is tied to throughput and latency, and these can be modeled through simulation. Moreover, simulation environments that can replicate the components of hybrid networks, software-defined exchanges, and other emerging technologies can also provide greater motivation to network operators to use these platforms.

## X. HYBRID SOFTWARE-DEFINED NETWORKS

For many network operators, the bar to implementing a ”clean slate" SDN network is simply too high [180]–[182]. Unlike greenfield deployments where networks are installed and configured in the absence of an existing network, companies and agencies are already heavily invested with their current infrastructure. Hence, implementing a network infrastructure to achieve SDN capabilities (e.g., a network operating system or NOS) is prohibitive. Couple the above issues with the lack of use cases for SDN or a defined path for migrating existing network infrastructures to SDN, and the possibility of an SDN-pure architecture is further compounded [32]. Thus, for these organizations, economics dictates that hybrid networks emerge well in advance of purely SDN architectures as network engineers incorporate SDN capable upgrades into equipment life-cycle iterations.

A couple of organizations working in Hybrid SDN include the IETF with their path computation element [14] and the OpenFlow Networking Foundation (ONF) [49] Hybrid Working Group (WG), who are evaluating various hybrid models such as ships in the night (SIN) where communication between legacy and OpenFlow control planes are denied interaction with no need to synchronize states between management and control planes [183]. Such an architecture provides isolation through per-VLAN and per-port segregation. Still much work is left to be completed.

An early example of a hybrid network includes Google’s Inter-Data Center WAN [7], [184]. Yet, while such networks enable automation, reduce overall equipment costs, and provide predictability once implemented [7], [8], [183], [184], they also require significant levels of research, engineering, labor-hours, and cost to accomplish. Yet, the average network operator may not have these resources available to them. As a result, network operators seeking to incorporate SDN into their network upgrades will likely follow a staged process [180]. As network operators begin this process, they enter into the realm of transitional networks consisting of both traditional network devices and SDN devices [180].

Because of the above circumstances, network operators will likely find their networks converging into one of four models: 1) topology-based, 2) service-based, 3) class-based, or 4) an integrated hybrid network [182]. For instance, network operators, who control enterprise networks that span several different regions or geographical locations, might be more inclined to incorporate a topology-based hybrid network. In this type of network, an SDN is set up in one region while a traditional network, consisting of legacy equipment, remains in other regions. A service-based or class-based model might also be appropriate for other network operators seeking to run both traditional network and SDN paradigms on their network concurrently. With a service-based hybrid model, the network operator may choose to use traffic engineering and load balancing at the network’s edge while using traditional network paradigms at the network’s core [182]. Using the service-based model also allows network operators to take advantage of edge-based, security features and programming frameworks offered for SDN, like [42], [140]–[142]. Then again, the network operator may be more concerned with only applying the SDN paradigm to TCP (port 80) traffic and using the traditional network paradigm for everything else. This would indicate the need for a class-based hybrid model. However, some service-based models and the class-based models require that switches allocate memory for both traditional network devices and SDN controllers, which does not make for a simpler data plane device. Moreover, the cost of converting all switches that support these two models may be prohibitive [182].

Since converting all switches to support SDN may not be a possibility, network operators may instead seek to deploy a partial service-based model that only replaces a minimum number of traditional network switches with OpenFlow switches in order to realize a majority of SDN functions without completely overhauling the network [180], [181]. Research by Levin et al. [180] has shown that over 80% of on an enterprise network can be operated as an SDN after upgrading less than 0.6% of available switches in networks consisting of 1500 or more switches while still meeting VLAN and flow table resource constraints. Additional use cases and procedural documentation for implementing this strategy is still needed to assist network operators with the placement of OpenFlow switches and SDN controller implementation on their networks. Finally, network operators might look to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



an integrated model that allows for SDN programming at the application level, but utilizes an interpreter to pass SDN application demands to the command line interface (CLI) of legacy switches. This method is a less intrusive approach, but complications with generating CLI commands for various proprietary network devices may prove too difficult.

Hybrid networks also carry their own set complications. Already, concerns over how to share control plane resources (e.g., ports, tables, and meters) have been indicated [183]. Yet, each of these models offer strengths and weaknesses for which network operators should be informed. Use cases to help network operators overcome their concerns with resilience, robustness, and scalability are also desired, and greater development is needed to achieve balance between traditional network protocols and SDN communications [14]. Similarly, options for interacting with operation, administration, and management (OAM) functions, legacy devices, and neighboring domains are needed [183]. The communication from legacy data planes to OpenFlow controllers and the communication between legacy and SDN control planes cannot be ignored either [183].

While industry and government may not be ready to fully embrace SDN-pure solutions for many years, targeted research focused on bridging the levee of legacy networks could further hasten the adoption of SDN by these organizations. By offering use cases and application assessments for hybrid SDN networks, as well as simulation models that demonstrate other benefits of partial SDN deployment, researchers can offer practical advice and procedures to network operators for incorporating SDN capabilities into their future plans for life-cycle replacement of legacy systems. Such use cases should demonstrate how to make OpenFlow-enabled switches fully compatible with current operational networks, or they may simply demonstrate how to use OpenFlow switches to complement the capabilities of legacy switches (e.g., flexibility for rule matching) [183]. In doing so, researchers are poised to highlight exactly how network operators might reduce cost and complexity, improve budget planning, introduce new capabilities, manage workloads, and improve reliability with dependable performance and predicted outages [185].

## XI. SOFTWARE-DEFINED EXCHANGE (SDX) / SOFTWARE-DEFINED INFRASTRUCTURE (SDI)

A Software-defined Exchange (SDX) allows participating organizations to introduce their policies to a centralized controller serving as an arbitrator for global policy for multiple domains and autonomous systems. With a global view, the SDX controller is able to introduce scaling techniques that can accommodate a large number of policies and participants [27]. As a result, SDX can resolve inter-domain routing problems that have long plagued exchange points by introducing new policies that handle packets at more granular level, while maintaining consistent BGP route advertisements [27]. Other benefits that SDX stands to contribute are prevention of policy violations, participant communication, and DoS attacks; forwarding optimizations for fast convergence, data offloading, middlebox traffic steering, and inbound traffic engineering; application specific peering; and remote control of BGP path selection and wide-area load balancing [186].

Already SDX is being implemented at various research and education networks [27], [187]–[190]. The range of SDX possibilities and use cases include Layer 2 Ethernet circuits, Layer 3 BGP policies, SDN multi-domain, and Software-defined Infrastructure (SDI) [191], [192]. However, agreement on what a Software-defined Exchange (SDX) is, what it should provide, and how SDX will provide it has not yet been reached [186], [193]. Questions posed include should it be a virtualized "meet-me" point where network operators bring their own compute and storage resources or just a place to exchange networking capabilities, like the already existing Internet exchange points (IXPs) [194].

To unify these definitions, the National Science Foundation (NSF) has proposed a distinction between SDI and SDX. An SDI takes advantage of the virtualization of computing and storage resources, such as SDN and software defined radio (SDR), to build more programmable and agile cyberinfrastructures, while an SDX enables the exchange of these resources across domains. Nonetheless, SDX seeks to employ SDN as a tool for overcoming the limitations experienced with traditional peering across Internet Exchange Points (IXPs). These limitations include those already associated with Border Gateway Protocol (BGP) and the lack of expressiveness afforded traffic policies [27]. On the operational side, another issue involves sending a request to a network operating center and the time required to obtain the connection [195].

SDX must also find sufficient compromise between multiple stakeholders consisting of content providers (e.g., Google, Netflix, etc.), "eyeballs" providers (e.g., Comcast, Verizon), and transit providers (e.g., AT&T and Internet 2) [27], [186]. Each of these stakeholders approach SDX with unique financial and political motivations. For instance, Internet Service Providers (ISPs) only want to share their external routing policies, but not their internal ones. On the other hand, research and education networks want to share compute, storage and networking resources. As a result, research in this field must not only consider the realm of the possible, but also the political and financial implications to network infrastructure stakeholders if their research is to be adapted.

The NSF currently sponsors a few projects in SDX [27], [188], [196]–[199]. One project includes leveraging an SDX to support large data flows from new telescopes in South America to supercompute centers in North America [188]. Others involve enhancing Software-defined Infrastructure (SDI) capabilities, improving dynamic resource allocation across multiple domains, and bridging SDN islands, topology exchange services, and multi-architecture frameworks [186], [187]. So, a wide range of applications exist within the SDX/SDI framework.

In regard to security, SDX offers the opportunity to prevent

or block policy violations, deny communication between participants, and block DoS attacks upstream [186]. Additionally, researchers are already seeking to incorporate the middlebox traffic steering, traffic offloading, and inbound traffic engineering features of SDN within SDX [27], [186], [188]. Accordingly, network operators already working at Internet Exchange Points may find SDX capabilities, such as application-specific peering and remote control, useful for influencing BGP path selection and performing wide-area load balancing [27].

Ongoing research in SDX includes reducing state space and control overhead, updating controllers, leveraging multiple tables and switches, and developing APIs for specifying policies [27], [186], [200]. In the realm of Internet Exchange Points, project Cardigan [190] is an SDX implementation utilizing RouteFlow-based, distributed routing and a mesh of OpenFlow switches, to mimic a single logical switch. To interconnect SDN islands, WE-Bridge offers a mechanism to allow various SDN administrative domains to peer and cooperate [201]. As a result, collaborations are forming to create large-scale federated testbeds across different continents such as the AtlanticWave-SDX [186], [188] between North and South America, and FELIX [202] between Europe and Japan. The next steps for SDX include operational deployments and the development of additional applications and distributed exchange points [53]. Such examples include deploying ubiquitously at the metro-level [203], providing accountability services at the SDX [204], and also Science DMZs.

Scalability of SDXes are also a concern, as well as limitations imposed by available hardware platforms [200]. Hardware capabilities are quickly catching up to network administrator's desires; however, implementation has thus far not been fully compliant at best, or incorrect at worst [205]. Moreover, the development of policies for facilitating cooperation amongst varies autonomous systems and peer groups are needed.

Another issue plaguing SDX and multi-domain SDN environments is policy enforcement across domains. Since SDN controllers in one domain cannot define nor monitor policies in other domains, network operators are unable ensure that their own policies are being enforced in domains external to their own. Hence, challenges to policy checking within SDX and multi-domain SDN environments presents a rich field of study for researchers seeking to better equip network operators. Solutions like LegoSDN [72], while intended to provide fault tolerance by sandboxing SDN applications, may also prove useful for providing limited control capabilities to network operators across domains–provided appropriate communication channels can be established. Consequently, some researchers have already made initial attempts to enforce policies for single domains using declarative languages [67], [206], [207] to express polices (i.e., forwarding rules), yet they do not consider secondary or tertiary domains. Thus, verifying that their origin policies are enforced in external domains remains a challenge. Likewise, other research for verifying SDN configurations, like VeriFlow [208], FatTire [209], FortNOX [210], and NetPlumber [211], seek to deconflict policy rules, validate legitimate rules, or detect misconfigurations; yet, they too do not consider validation of policies in external domains.

This presents a problem when database servers are located in external domains as network operators cannot be sure that their policies are being enforced for the said server. For instance, network operators may want to ensure that network policies are blocking all flows to the database server except for those originating from their IT department. Hence, [212] offers an SAT-based, solution, AudIt, which attempts to solve the foreign controller verification problem by auditing network policies across multiple domains to determine whether origin policies are enforced. Still, AudIt [212] can only verify whether a policy is enforced, and measures for enforcing polices across multiple-domains without circumventing each domain's origin policies are still largely absent. AudIt also requires that OpenFlow extensions be added to participating switches. Likewise, performing policy verification and enforcement across domains without creating additional security risks is yet another field of research in need of development.

Moreover, to support agile end-to-end management and orchestration of resources and services in multi-operator environments, a common marketplace for resource exchange is required. Further hindering agile management and orchestration are multi-operator services, which are protected by legal contracts and take long periods of time for parties to define and reach agreement. Hence, Griffioen et al. [204] proposed a coin-operated SDX for IXPs that defines an economic plane to tie routing policies to economic relationships of cost and benefit between peers. Their initial proof-of-concept uses the concept of a "coin" for authentication and charge of flow rule requests between IXP peers. However, if the coin concept if not implemented correctly, many issues related to digital currencies (e.g., double spending) may degrade trust in the SDX.

## XII. EMERGING TECHNOLOGIES

As we initially indicated, the areas of research comprising SDN are indeed numerous. Likewise, the demand for applications that enhance productivity and leisure will continue to grow as dependence on augmented reality applications and many others become the new norm. As a result, SDN is steadily proliferating throughout emerging technologies (e.g., IoT, ICN, Wireless, 5G, etc.). Within all these areas, SDN offers the ability to provide fine-grained, QoS-aware resource allocation for singular flows while also addressing changing traffic patterns via dynamic network reconfiguration. Hence, we attempt to offer a brief overview of these emerging technologies along with further research opportunities serving to advance SDN.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



### A. INTERNET OF THINGS (IOT)

The future seems to indicate an inter-networking of nearly everything, including traditional end devices (e.g., laptops and smartphones), home appliances (e.g., thermostats and refrigerators), industrial systems, people, and much more. As a result, anywhere from 20 billion to 50 billion devices are projected to connect to the Internet by 2020 [213], and many network operators are also trying to adapt their networks for what is now being dubbed the Internet of Things (IoT). This IoT paradigm may potentially revolutionize the way people live and work via a wealth of new services; however, these services must run atop an enormous variety of heterogeneous devices requiring a greater diversity of communication requirements and application domains [214]. Security standards and mandates are also sufficiently lacking for these devices [215]. Consequently, this heterogeneity and lack of security, coupled with large-scale and latency sensitive applications for numerous devices, challenges network operator attempts to fully realize the IoT vision.

Fortunately, the SDN paradigm can potentially be applied to IoT as an enabler for numerous SDN-based IoT applications. Yet, these applications also come with their own challenges. For instance, SDN requires greater abstractions to achieve the required levels of service and security needed to fully augment IoT infrastructures. Moreover, it may also require integration with virtualized components. For instance, a natural application to address the heterogeneous QoS requirements of IoT is to include network slicing [213]. Additionally, IoT device security may require the incorporation of NFV technologies to aid SDN controllers with monitoring and enforcing policies on IoT network flows to improve security [77]. Consequently, many opportunities for advancing SDN-enabled IoT applications exist.

#### 1) Wireless Sensor and Actuator Networks

While SDN enables the programming of core network devices, allowing for the injection of routing logic [216], it has yet to offer a solution for guaranteeing correctness and temporal coherence for software controlled actuators and other devices [217]. Already, researchers are looking for novel ways to employ OpenFlow technology to assist network operators with wireless networks. Moreover, substantial research on interconnecting Wireless Sensor Networks (WSNs) into wider IoT frameworks by leveraging SDN and virtualization has already been completed [213], [218]. Yet, other goals include improving reachability, sharing of resources, and improving scalability [219]. Within this framework, Wireless Sensor and Actuator Networks (WSANs) have arisen, and software-defined and software-controlled WSANs are emerging as research topics for application development. Still, there remains a question as to the extent wireless sensor and actuator network limitations will affect the SDN paradigm [217]. Furthermore, applications in this arena have yet to fully surface.

#### 2) IoT in Urban Environments

Urban environments, particularly their transportation, utilities, and law enforcement services, stand to benefit from IoT. However, an interesting argument of the works covered in [213] is that IoT urban deployments should allow for the same set of sensor nodes to support multiple applications from multiple developers across the same shared physical infrastructure via software only. This makes SDN an ideal candidate, since developers can exploit its Northbound APIs to orchestrate an IoT network. Consequently, their proposed network consists of three layers, which are much akin to current SDN networks:

1) **Physical layer** This layer contains physical network devices (e.g., sensors, smartphones capable of sensing, base stations/access points, and the gateways connecting to the network's backbone). As with the SDN paradigm, the SDN-IoT architecture nodes in this layer lack intelligence and relies on its control layer for decision making.
2) **Control layer** Residing between the Physical layer and the Application layer, it manages the devices in the physical layer and offers developer APIs to the application layer. For urban sensing, it also offers aggregated data, network transmission, and processing. What's more, the SDN controller resides in this layer ensuring device sharing and QoS-aware routing of the generated data from the core network towards its eventual clients.
3) **Application layer** Developers use the abstractions in this layer to build IoT applications. As a result, the underlying layers and physical infrastructure are hidden from the developer as they work to create applications for IoT.

Of course, this framework is loaded with challenges. Some include translation of application requirements as they relate to QoS and geographical sensor location, inner sensor node configurations, sharing of sensors amongst competing applications, transmitting optimized and QoS-aware data flows towards end-servers, and efficient distribution of data for cloud processing [213]. Additionally, fully centralized SDN controller architectures do not adequately address the different access networks expected to comprise the urban-scale IoT mobile multi-networks, such as LTE, WiFi, or ZigBee [213]. Hence, a distributed scheme is called for. Likewise, mobility is a consideration as IoT devices are expected to roam from one access point to another [220]. Scalability issues are also addressed in [220] with a distributed hashing algorithm for assigning IoT devices to a respective SDN controller. It also serves to reassign IoT devices to access points to manage loads and mobility. However, work to optimize flow scheduling for the backbone network is still not fully developed [213].

#### 3) VANETs

Another aspect of an IoT infrastructure is the Vehicular Ad Hoc Network (VANET) [221]. Within this context, vehicles

communicate to each other (V2V) in an ad hoc manner and with fixed infrastructure consisting of roadside transceivers or cellular base stations. In this setting, an SDN controller can perform routing actions while vehicles and roadside units serve as SDN switches. We mention this work to provide another aspect of emerging technologies with regard to SDN-IoT infrastructure, and refer the reader to [213], [221], for a more complete overview.

#### 4) BYOD

The bandwidth-intensive nature of bring your own device (BYOD) initiatives also falls within the context of IoT, serving as yet another driver for SDN adoption. For instance, Hong et al. [222] propose a fine-grained network security framework for network management and policy enforcement on mobile apps and devices in enterprise networks, by the use of virtual switches and containers to extend SDN capabilities to the end host.

Still, as the authors point out, SDN-based solutions on the client side are still subject to various challenges [222]. These include system circumvention by disabling context functionality (e.g., GPS, PBS-DROID, etc.), portability between a multitude of devices, protocol and interference coverage (e.g., current solutions only support TCP), controller scalability (which is a universal concern of SDN applications), and SDN-based attacks (see §VII).

#### 5) IoT Security

Another aspect of IoT that concerns many experts is the lack of security standards for front-end IoT devices as they proliferate and increasing connect to one another and the Internet [215]. These same experts also point out that manufacturers of these devices do so with a focus on function and cost–a focus of their consumers. However, without standards or government mandates, manufacturers have little motivation to raise costs and secure these devices. The result being a vastly growing attack surface. With the security of these devices in question, network operators are further challenged to ensure that these devices cannot somehow be leveraged for malicious purposes within their own networks.

To this end, SDN-based frameworks are needed to monitor network flows, detect anomalies, and automatically prevent malicious activity on organizational networks. Much of this activity is best detected along the network's edge as has been observed by [4], [141], [142]. Opportunities for doing so, however, remain widely unexplored.

### B. INFORMATION-CENTRIC NETWORKING (ICN).

Network operators are also challenged to address new routing protocols aimed at improving efficiency of content delivery and content availability for their clients [2], [58]. One such architecture is Information-Centric Networking (ICN) or Content-Centric Networking (CCN) where packets are routed based on desired content instead of traditional, location-based, addressing [58]. Within this context, desired content is moved from its home server to a location (e.g., an information cache) closer to the client desiring it.

Past projects [223]–[225] have already demonstrated various SDN applications or have developed prototype ICNs, which demonstrate how SDN applications can positively impact the roll-out of ICN solutions. NFV components–essentially virtualized software instances deployed in Virtual Machines (VMs)–can also be chained together and managed through an SDN controller to enhance end-to-end content distribution.

For instance, the concept of Information-Centric Network (ICN) is combined with virtualization in [213], [226] to provide network slicing and node caching within a wireless network. The result being that desired content is closer to the clients requesting it. To do so, they offer three slicing paradigms, consisting of Network-level slicing, Flow-level slicing, and Content-level slicing. The more interesting of these is their Content-level slicing where the content cache is virtualized and content is divided into multiple slices for different users. Accordingly, an SDN controller is suggested as a means for allowing network operators to optimize mappings between available physical resources and the virtual resources granting services. However, as observed by [213], their work lacks any methods or algorithms for actually achieving this optimization.

Similarly, NDNFlow [213], [227] utilizes Named Data Networking (NDN) and SDN to setup and facilitate ICN networks. However, to avoid modification of the OpenFlow specifications, they choose to incorporate separate, parallel ICN layer using a separate controller module. This approach allows ICN flows to be handled separately from regular IP flows. However, their method also requires that software plug-ins be installed on end elements to render them ICN capable. Furthermore, SDN solutions for ICNs still require the creation of more adaptive northbound APIs to better support their applications. Likewise, an expansion of the OpenFlow protocol to support customized header matching may also be required to better enable traffic engineering that supports ICNs [2].

### C. FIFTH GENERATION (5G)

As with IoT, and perhaps because of IoT, the explosion of mobile traffic along with the combination of highly virtualized environments are exposing limitations in current telco networks and the need for highly dynamic and more scalable networks. For telcos, the development of services has progressed with featured services (e.g., 2G-voice, 3G-data, and 4G-speed). For 5G, many researchers believe its defining characteristic will be services at scale, offering lower application latency [228]. However, [229] see two main challenges for 5G networks: 1) a reliable connection despite increased data traffic due to IoT and 2) maximum end-to-end delay guarantees for real-time applications. Accordingly, future 5G networks can also expect extreme traffic volumes over both its fronthaul and backhaul [213]. Likewise, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



vast amount of usage scenarios coupled with heterogeneous devices and requirements still offers many hurdles [213].

Accordingly, many researchers see countless use case for 5G that includes broadband access everywhere (e.g., 50+ MBPS everywhere), broadband access in dense areas (e.g., pervasive video), higher user mobility (e.g., high speed train), massive Internet of Things (e.g., sensor networks), extreme real-time communications (e.g., tactile Internet), lifeline communications (e.g., natural disaster), ultra-reliable communications (e.g., e-health services), broadcast-like services (e.g., broadcast services), and many others yet unimagined today [230]. For instance, much of the smart home sensory data from devices, like smart meters, temperature sensors, security cameras, multimedia sensors, etc., are processed locally; however, new technologies, like virtual reality, augmented reality, semantic recommendations, tactile Internet, real-time pattern recognition, and a growing number of others, require that greater amounts of sensory data be processed in virtualized servers in remote locations with outcomes made available to the user [229]. In such cases, the challenge lies with timely results that require near-real-time guarantees.

Hence, research within this context has the potential to shape much of network landscape for network operators in coming years. Additionally, the combination of SDN's centralized controller and virtualization technologies to implement various network functions is seen as a key enabler for future 5G networks [231]. For example, SDN-based architectures can provide dynamic topology reconfiguration, while NFV can help address local content sharing challenges by providing storage and processing resources [229]. As a result of SDN and virtualization technologies, new technology enablers, like self-organizing networking (SON), the cloud radio access network (C-RAN), and mobile edge computing (MEC) may soon be fully realized and better support content specific topologies for information-centric networking (ICN), instead of an IP-based topology as discussed in [229].

SoftAir [232] represents one framework that combines SDN, NFV, and NV into a complete 5G cellular system. Some of its key aspects include high network flexibility from moving functions (e.g., mobility management, QoS routing, and billing) to the cloud and network slicing to dynamically allocate isolated subnets to different network entities to provide customized services [213], [232]. Another key aspect is a central SDN-enabled cloud orchestration, which includes a mobility aware traffic management module, a distributed traffic classification function, and a resource-efficient network virtualization module [213], [232]. Additional simplifications provided by SoftAir are location management in terms of less signaling overhead and rerouting in case of handoff using a centralized controller to exploit combinations of cellular and WiFi access networks and a global network topology to choose new best paths [213], [232]. As described, Softair forms a complete system-level design that could encompass every aspect of 5G; however, it remains to be determined if the performance of such a system and its algorithms will prove commercially viable.

As stated previously, the capacity of certain coverage areas are seen as a significant challenge for 5G. Options for dealing with increased capacity include using more spectrum bands, deploying new technologies to improve spectral efficiency (Hertz rate), increasing access node numbers, or using device-to-device communication techniques [229]. As a result of these capacity enhancements, network densification is seen as the dominant theme for 5G's wireless evolution [229], [233]. Yet, network densification also creates its own challenges for network management. For instance, as small cells increase, the distance between channels using the same frequency will decrease while the edges for cell coverage areas increase. These changes will lead to more interference between cells as well as more frequent handovers. Likewise, the increased number of cells will lead to increased energy consumption. To address these issues, [229] identify several areas where SDN can assist other technologies in mitigating these challenges–as listed below.

- Spectral efficiency: dynamic frequency resource allocation and centralized transmission power control
- Mobility Management: coordination and optimization of handovers, predictive learning methods, and storage of mobility data
- Energy efficiency: topology manager for active cell minimization and moving features like spectrum sensing to data centers to remove energy costs from cells
- Cloud-IoT integration: centralized utilization of frequency resources and consideration for QoS/QoE of IoT-Cloud communication

Other research [234] has identified opportunities for using SDN to share the licensed and unlicensed bands of spectral resources to achieve gigabit data rates in 5G. Unlicensed bands might include industrial, scientific, and medical (ISM) band, visible light communication (VLC), and millimeter-wave (mm-Wave bands) [234]. This functionality can also enable device-to-device (D2D) or machine-to-machine (M2M) communications. To do so, in [234], the authors attempt to balance task distribution between base stations (BS) and the SDN controller. Additionally, they argue that the spectrum can be better utilized by using data-base-assisted spectrum management over spectrum sensing, which they identify as a key challenge to spectrum sharing due to the uncertainties imposed by the wireless medium. They identify other key challenges to spectrum sharing as decision making and admission control. With the SDN controller orchestrating the back-end network and BSs comprising the front-end, the authors present a framework and an efficient resource management algorithm for future 5G networks. Simulations also indicate that potential performance gains in access and reduced interference can be achieved. Still, real world applications and metrics for this framework have yet to surface.

SDN also offers unique security features for future 5G networks. For instance, [235] offer privacy protection for 5G networks by introducing an SDN-based efficient authentica-

tion handover and privacy protection solution. Their work suggests that SDN-enabled security solutions are particularly relevant for delay-constrained 5G networks.

The potential contributions of SDN to 5G are potentially numerous; yet, some concerns must also be addressed. One concern for SDN, due to its logically centralized controller, includes scalability, which authors argue is still not sufficiently addressed for 5G [229], [234]. Another concern is latency, especially as results of IoT devices must be rendered in real time [234]. Additionally, if spectrum sharing is considered, compelling business models and safe guards are needed to entice organizations to cooperate.

### D. SOFTWARE-DEFINED WIRELESS NETWORKS (SDWN)

As SDN and OpenFlow developed, they did so with infrastructure networks (specifically, wired networks) in mind [236]. As a consequence, the path to wireless platforms is a challenging one. As we discussed in §IX, even SDN wireless simulators have only recently been developed. Yet, wireless provides campus, industry, and government networks tremendous flexibility and are highly desired.

Wired connections impose costs as new lines are installed and maintained, and each new line increases the network's complexity and diameter limitations [236]. However, wired connections still possess enough flexibility to overcome these challenges. Wireless, however, imposes limits that are harder to overcome and users are often subjected to rate limitations [236]. Wireless networks also have specific requirements like dynamic channel configuration, mobility management, and rapid client re-association [237].

To successfully utilize SDN for wireless, network operators must be able to limit interactions between defined slices, and their devices must communicate status information to the controller [236]. Note that a network slice is simply a mechanism for dividing the network infrastructure into disparate partitions (or subnetworks) so multiple instances can coexist [237]. Unfortunately, these operations are far from simple. For instance, slicing requires the isolation of communication channels so a FlowVisor application (i.e., an OpenFlow-based proxy layer that creates slices based on parameters like flow-space, bandwidth, and CPU load [237]) can offer different coordinators a list of non-interfering subnetworks [236]. A wireless SDN must therefore manage a limited number of independent channels while maintaining a global view of the network to avoid link interference and reuse geographical channels [236]. Regarding status updates, wireless requires that controllers receive more than CPU load or available memory. They also require link information (e.g. load, delay, loss, rate, and stability) and topology discovery (which includes identification of local access points) [236].

Since many wireless users are also mobile, wireless SDNs must be able to affectively coordinate the handoff of wireless devices from one access point to another. So, solving issues, like separation of data channels, slicing, topology discovery, channel estimation, and interference management, are key to offering clients a reliable and robust wireless SDN. These issues are compounded by that fact that many networks consist of hundreds of access points [159]. Accordingly, network operators need tools for assessing the scalability of such implementations before deploying them. To that end, tools, which are notably missing, are needed. Many such models or simulators [154], [160], [238] allow researchers to emulate SDN or wireless, but not both. Only recently, has open source tools [159], [161], capable of modeling wireless SDNs, been offered, while a commercial simulation tool, EstiNet [162], was introduced in 2013.

If the challenges of wireless SDN can be overcome, then there are also a number of opportunities that will greatly advantage network operators. Those include improved end-user connectivity and QoS, multi-network planning, enhanced security and user localization [236]. Other benefits include freedom from expensive commercial wireless-management products in addition to flexible management, traffic isolation, and link-layer mobility management [159]. However, these opportunities still need to come in the form of feasible implementations that network operators can deploy on top of their existing systems. As with other SDN applications, use cases and best practices are desired.

### E. SOFTWARE-DEFINED RADIO ACCESS NETWORKS (SDRAN)

Radio access networks (RAN) make up an important part of cellular networks, providing wide-area wireless connectivity to mobile devices. However, managing a limited spectrum to maintain connectivity for a growing number of mobile devices is one challenge that researchers believe SDN can help address. With SDN, researchers hope to better allocate resources, setup handovers, balance loads, and manage interference [239]. This trend has also been growing over the past several years. From [2], some interesting work that could be done in this field include 1) collecting and analyzing the quality measurement data from the base station and mobile terminals, and correspondingly adjusting the settings of the base station and 2) developing various SDN-based scheduling mechanisms for managing uplink and downlink flows. In the case of SoftRAN [234], [239], a virtual big base station (BS), comprised of a centralized controller and physical BS, implements an architecture for coordinating radio resource through the software defined control plane of the RAN.

Cai et al. [240] advance the application of SDN and NFV for RANs by observing that the control plane and data plane of a RAN requires a significant amount of modification to implement device-to-device (D2D) communications. However, integration of D2D communications can be greatly facilitated by SDN and NFV. Within the context of an SDN and NFV enabled architecture, they address the imperfectness of network state information (e.g., channel state information (CSI) and queuing state information (QSI)) in virtual wireless networks for D2D communications. Accordingly, they develop a resource sharing scheme and demonstrate through simulation that this architecture can achieve considerable

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



performance gains in both user utility and system throughput under common network conditions.

### F. DISCUSSION OF SDN-BASED OPEN RESEARCH IN EMERGING TECHNOLOGIES

As has been pointed out in [213], [229], further optimization of SDN and OpenFlow to accommodate the peculiarities of IoT are still needed. For instance, the growing number of participating devices leads to ever-growing flow tables on SDN switches. Likewise, with devices continuously entering and leaving subnetworks, enabling and disabling flows in a rapid manner creates substantial overhead requiring further optimization. Hence, further study of temporal behaviors and requirements of IoT devices can further benefit this framework.

Additionally, the extreme heterogeneity of devices participating in IoT infrastructure and their requirements are other points of consideration [213]. In this context, smart vehicle applications might require almost zero latency due to their dependency on other vehicles. Yet, IoT sensors in an industrial plant might allow for minimal packet loss, while video surveillance applications could potentiality tolerate both latency and modest packet loss–but with substantially higher bandwidth requirements. Consequently, having SDN offer smart routing and scheduling solutions in conjunction with virtualization to form network slices could better isolate such IoT use-cases having conflicting requirements.

Cellular technology is also expected to serve as a critical enabler for IoT, and SDN and NFV are expected to be key factors in handling the vast increase of data traversing both access and core networks [213]. Extended automation, QoS-aware differentiation of IoT traffic classes, and data collection/analysis to enable network optimization through SDN are also required to advance IoT applications. Hence, it is important that 5G architecture designs prepare to handle the onslaught of data produced by growing IoT applications. Likewise, the realization of IoT on a large scale requires that it be fully integrated with cloud computing [229]. Additionally, due to limited energy, storage, and compute, the massive data produced by these devices may only be useful if IoT and cloud resources are deployed together [229].

Security is yet another advantage that SDN potentially brings to emerging technologists, assuming the SDN controller can be protected from exploitation. The challenge here lies in the development of time-sensitive and mission critical applications capable of detecting and mitigating specific security threats. As we discussed in §VII-B, the network's edge is the least complex way to capitalize on device state for detecting various threats. SDN also enables the creation of complex rules for device-access. As a result, the complexity of multi-network management for hundreds of different devices, having different permission levels for different clients, can be greatly reduced for network operators. Additionally, network slicing via virtualization can help isolate sensitive traffic flows to ensure the data's confidentiality and integrity. As previously alluded to, the development and standardization of such security protocols, interfaces, and applications specifically tailored to emerging technologies (i.e., IoT, 5G) are still open areas of research.

New machine learning techniques can also be used in conjunction SDN's centralized network management and data collection to make networks more intelligent and capable of responding to client needs. Likewise, learning algorithms can allow traffic characteristics to influence real-time decisions. Hence, the development of customized machine learning algorithms–merging networking and artificial intelligence paradigms–to handle traffic flow across these emerging technologies is also expected to be an emerging research field [213].

## XIII. DISCUSSION

We have observed that SDN's key innovations hinge on its ability to separate the control and data planes and provide centralized control and programmability over the network. Likewise, SDN, NFV, and NV combinations bring significant benefits to 5G, IoT, and other developing network infrastructures. With SDN, these innovations can be further complemented once standard southbound, northbound, and east-west APIs are accepted. As [16] observes, even carriers who have historically been hesitant to embrace new technologies that might disrupt network services are turning to SDN-NFV technologies to reduce CAPEX and better monetize their services. Additionally, due to the complementary nature of NFV and SDN technologies to increase flexibility, support scalability, and speed up the introduction of new services, we are already seeing a convergence of these technologies, enhancing cloud management platforms and network service orchestration platforms [2], [19]. Additionally, SDN, NFV, and Cloud are all expected to play a significant role in future development of 5G services.

Likely, both SDN and NFV will continue to drive innovation in telecommunications, networking and enterprise data centers [2]. Both technologies serve to decrease costs, increase flexibility, accelerate the introduction of new services, and support scalability. However, SDN when applied to NFV may provide network operators with a better solution for dynamic resource management and intelligent service orchestration [2]. The challenge that still faces this technology is how to seamlessly integrate virtual network functions into existing SDN and cloud infrastructures to support multi-tenancy [2]. Other issues for network operators include ensuring that their SDN controllers provide them with the features they need (e.g., flow mapping applications, east-west bound communication, etc.).

Perhaps the greatest contribution SDN provides to network service providers (NSP) is service chaining. By steering traffic between network functions, SDN has provided for greater flexibility in the placement of network functions (or service components). However, NSPs currently do not possess the ability to orchestrate the placement of network functions in their networks in an optimal way. Hence, future research opportunities with service function orchestration are

numerous.

Network operators must consider certain factors when deploying SDN and NFV solutions, including: 1) which network functions to virtualize, 2) return on investment, 3) interoperability, 4) scalability and elasticity issues, 5) resource management, and 6) APIs allowing ease of management [2]. However, certain elements of these technologies can still slow down adoption [16]. For instance, one key element is a shortage of adequate skill sets can hinder an organization's ability to integrate an SDN architecture with its network. Partly, this is because of the push towards automated and highly programmable networks represents a significant paradigm shift in skill sets that stray from the highly specialized network engineer. Hence, the need for training and certification. Another element is the complexity of solutions. According to [16], the complexity of a full roll out is much higher than originally anticipated. As a result, full SDN deployments represent only a small fraction of what earlier predictions anticipated.

Integrating with real-world production systems is yet another challenge to SDN deployments. As we discussed in §X, many organizations do not have the luxury of greenfield deployments. Instead, they must integrate new SDN technologies with existing network infrastructures. This drives a pertinent question within such hybrid solutions. How will orchestration for the data plane be handled for such networks. As observed by [16], interoperability between SDN and traditional network architectures can be trying, even for more experienced network operators. As a result, SDN has much room for improvement, and network operators are not placed in ideal positions for improving it. Consequently, the majority of effort needed to make SDN more deployable will fall to the research community (industry, government, and academia) and standardization bodies. The main challenges that we identified as hindering the adoption of SDN by network operators include the lack of an intuitive APIs for the north-bound interface, the lack of standardization and certification for such interfaces, and the lack of security offered in current SDN architecture.

Even now state-of-the-art security operations rely on network operators to interpret high-level policy documents and then implement them through low level interfaces while also manually implementing exceptions to accommodate various academic or operational requirements. They must also interpret a variety of alerts and reports generated by various security mechanisms (often middleboxes) and still be able to handle actual security events. Hence, network operators and service providers stand to benefit from the intelligent control of systems and the ability to orchestrate (or program) thousands of network devices. In addition, this paradigm makes network operators better suited to provide QoS guarantees for services like voice, video, and data while concurrently offering security through service isolation.

SDN indeed offers incredible opportunities, yet it also possesses equitable challenges for network operators. With a centralized controller, network operators can easily implement control changes by altering the control program [104]. Unfortunately, the absence of a standardized application interface for network operators means that creating control programs is not as simple as it needs to be. However, once intuitive abstractions become the norm for SDN, network operators will benefit from the ability to easily orchestrate a large number of network devices.

Provided a simple API can be offered, network operators will develop policies and dynamically implement them as requirements dictate via the control program. In a traditional control plane, which is completely distributed, such a feat would be extremely difficult and time consuming. Hence, it is the concept of programming the control plane that makes SDN such a powerful option for network operators; yet, we, again, argue that the abstractions offered by APIs must be intuitive. API intuition is a huge challenge for researchers who must work through a lack of documentation and essentially peel back the layers of abstraction when it comes to SDN programming languages in order to implement their own designs. We cannot expect this to be the norm for network operators, nor should we.

Network operators still need better ways to articulate the traffic they wish to monitor at higher levels of abstractions than is provided by current mechanisms [70]. Researchers must also maintain awareness that network operators need more than routing from SDN. Network operators make their living providing services–SDN is merely a platform to provide those services. Additionally, unless network operators are a part of an organization with a significant research budget for engineering and integration, they have limited support with testing SDN alongside their existing traditional networks. Thus, the research community must drive the further development of the SDN paradigm with additional mechanisms and applications for deploying these services if the adoption of SDN is to be hastened.

Thus, simulation can assist network operators to determine if SDN is the right solution for their network, since they can quickly ascertain whether their network possesses the flexibility to adapt new technologies, protocols, systems, and security features. Likewise, models offer network engineers a new means of explaining the impact of a network change prior to its implementation. More importantly, modeling can potentially provide network operators with pictures, graphs, and statistics needed to portray the advantages of SDN and convince their managers, change control boards, and budget review boards that SDN is the best solution. The result would be a better understanding of potential risks and, hopefully, greater approval rates for network changes.

Researchers must also consider the time-line that goes beyond 5th generation (5G)–the network development period after 2020 [229]. SDN is expected to serve as a technological enabler that is thoroughly integrated with NFV and Cloud technologies to offer self-organizing networks (SON), cloud radio access network (C-RAN), mobile edge computing (MEC), and content-specific topologies. Likewise, SDN can potentially revolutionize core network, backhaul, and fron-

thaul designs of next generation wireless architecture, while also serving as a key enabler for Radio Access Networks (RAN) and Cloud Radio Access Networks (CRAN) [229]. Without question, smart network functions are needed to make the radio technologies support more flexible networks. Hence, dynamic management enablers, such as SDN, and virtualization technologies are needed to meet these requirements.

Finally, we note that the surveys in this work focus on SDN deployments in the United States. While some of these issue may hold true in other countries, such findings are not contained in this work. Hence a focused international SDN deployment survey would also make for a valuable research contribution.

## XIV. CONCLUSION

SDN seeks to apply modularity to network control and gives network operators the opportunity to dynamically configure the control plane. Consequently, SDN's uses have found their way into virtualization, security, wireless, and numerous other technologies where greater opportunities exist for SDN's advancement. In this work, we presented a thorough study of SDN, its enablers, and current research challenges and opportunities. We began this study with the state of SDN as it relates to recent market reports and surveys where we identified some of the primary drivers and concerns impacting SDN's advancement. Having identified specific requirements, we then systematically explored opportunities for advancement with SDN's control plane and data plane, its interoperability with virtualization technologies, security impacts and concerns, modeling and simulation technologies, hybrid networks, software-defined infrastructures, and other emerging technologies.

Within the forwarding plane, SDN has already solved many problems including cost, management, multi-tenancy, and high entry-level barriers that stifle innovation. The ability to simultaneously program (or orchestrate) a large number of network devices makes SDN a powerful asset for future network operators. Yet, transition strategies must be considered, user-friendly applications must be developed, and standards must be agreed upon. Given the investment that campus, industry, and government organizations have already committed to SDN, we are certain that our future networks will be far more programmable than they are today. However, the key to this achievement is ensuring that network engineers and operators are provided a standardized, application rich, architecture to incrementally add to their existing network or setup as a greenfield deployment.

With regard to the future of SDN, common factors across all research areas, including, latency, jitter, flow rate, redundancy, reliability, security, cost, and availability, must be addressed [32]. As a result, network operators can still expect to see several revisions to SDN's architecture, interfaces, and applications prior to realizing an optimal set of capabilities. Accordingly, this paper identified and discussed some of the main challenges hindering the adoption of SDN by network operators. Many of which were repeatedly highlighted: the lack of an intuitive API for the north-bound interface, the lack of standardization of such interfaces, and the lack of security provided by the current SDN architecture. We also argued that the research community must continue to drive further innovations in SDN, as network operators may lack the resources and support needed to embrace SDN in their own infrastructures.

This work also provided a current snapshot of SDN adoption across government, industry, and campus networks as determined by various market reports and surveys. It may also serve as a reference point to steer future efforts of both researchers and industry practitioners seeking to contribute to the advancement of SDN–specifically, with regard to the numerous applications (both current and envisioned) within the context of emerging technologies. Moreover, this work addressed immediate research needs for enhancing traditional networks with SDN.

## REFERENCES

[1] "OpenDayLight." [Online]. Available: https://www.opendaylight.org/
[2] S. K. Rao, "SDN and Its Use-Cases–NV and NFV," Network, vol. 2, p. H6, 2014.
[3] SDxCentral, "The Future of Network Virtualization and SDN Controllers," Market Report, 2016.
[4] M. Casado et al., "Fabric: A Retrospective on Evolving SDN," in Proceedings of the First Workshop on Hot Topics in Software Defined Networking. ACM, 2012, pp. 85–90.
[5] ONF, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation White Paper, 2012.
[6] M. Bindhu and G. Ramesh, "The Journey to SDN: A Peek into the History of Programmable Networks," International Journal of Computer Science and Engineering Communications., vol. 2, no. 5, pp. 500–506, 2014.
[7] S. Jain et al., "B4: Experience with a Globally-Deployed Software Defined WAN," in ACM SIGCOMM Computer Communication Review, vol. 43, no. 4. ACM, 2013, pp. 3–14.
[8] C. Hong et al., "Achieving High Utilization with Software-Driven WAN," in ACM SIGCOMM Computer Communication Review, vol. 43, no. 4. ACM, 2013, pp. 15–26.
[9] S. Marek, "Verizon Executive Calls for SDN Standardization," Fierce Telecon, May 2015. [Online]. Available: http://www.fiercetelecom.com/story/verizon-executive-calls-sdn-standardization/2015-05-12
[10] A. Irei, "Verizon Offers Managed SD-WAN, with Cisco IWAN Technology," TechTarget, September 2015. [Online]. Available: http://searchsdn.techtarget.com/news/4500253097/Verizon-offers-managed-SD-WAN-with-Cisco-IWAN-technology
[11] NSF, DoE, and NCO, "Program Review: Operationalization of Software-Defined Networks," December 2013. [Online]. Available: https://www.nitrd.gov/pubs/SDN_Program_Review_Report_2013.pdf
[12] B. Roach, "3 Reasons Software-Defined Networking is Streamlining DOD IT," Defense Systems, April 2015. [Online]. Available: http://defensesystems.com/articles/2015/04/14/comment-sdn-software-defined-networking-dod.aspx.
[13] 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.
[14] S. Sezer et al., "Are We Ready for SDN? Implementation Challenges for Software-Defined Networks," Communications Magazine, IEEE, vol. 51, no. 7, pp. 36–43, 2013.
[15] D. Kreutz et al., "Software-Defined Networking: A Comprehensive Survey," proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, 2015.
[16] SDxCentral, "SDxCentral SDN and NFV Market Size Report," 2015. [Online]. Available: https://www.sdxcentral.com/reports/sdn-nfv-market-size-forecast-report-2015/
[17] PiperJaffray, "2015 Piper Jaffray CIO Survey," January 2015. [Online]. Available: https://piper2.bluematrix.com/sellside/EmailDocViewer?encrypt=7856c68e-3f1a-4ce9-a7e7-99fe25145cd9\&mime=pdf

[18] K. Marco, "InformationWeek 2014 State of the Data Center Survey," June 2014. [Online]. Available: http://reports.informationweek.com/abstract/6/12525/Data-Center/research-2014-state-of-the-data-center.html

[19] SevOne, "[White Paper] State of SDN: 6 Reasons Why SDN Is On The Rise," 2016. [Online]. Available: http://info.sevone.com/rs/505-CNP-948/images/SDN%20Whitepaper.pdf

[20] MeriTalk and Juniper Networks and General Dynamics, "Automation to Support Government Imperatives: Dawn of SDN," Tech. Rep., 2015. [Online]. Available: https://www.meritalk.com/study/dawn-of-sdn/

[21] Juniper Networks, Inc. and Wakefield Research, "Readiness, Benefits, and Barriers: an SDN Progress Report," Tech. Rep., 2014. [Online]. Available: https://www.usebackpack.com/resources/7178/download?1451715494

[22] CIO/G-6, "Shaping the Army Network 2025-2040," March 2016. [Online]. Available: http://ciog6.army.mil/Portals/1/Shaping%20the%20Army%20Network%202025-2040.pdf

[23] E. I. S. Group, "Network Function Virtualisation," http://portal.etsi.org/portal/server.pt/community/NFV/367, October 2015.

[24] R. Mijumbi et al., "Management and Orchestration Challenges in Network Functions Virtualization," IEEE Communications Magazine, vol. 54, no. 1, pp. 98–105, 2016.

[25] G. Finnie, "Policy Control and SDN: A Perfect Match." [Online]. Available: http://www.informationweek.com/pdf_whitepapers/approved/1370538516_hr_sandvine_policy.pdf

[26] E. Maini, "Orchestration of Logical Resources in Software Defined Infrastructures," 2015.

[27] A. Gupta et al., "SDX: A Software Defined Internet Exchange," in Proceedings of the 2014 ACM conference on SIGCOMM. ACM, 2014, pp. 551–562.

[28] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting Security Attacks in Software-Defined Networks," in Proceedings of the 2015 Network and Distributed System Security (NDSS) Symposium, 2015.

[29] ONF, "ONF Certified SDN Engineer (OCSE)," 2017. [Online]. Available: https://www.opennetworking.org/skills-engineer\#prerequisiteknowledge

[30] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, Attributes, and Use Cases: A Compass for SDN," Communications Magazine, IEEE, vol. 52, no. 6, pp. 210–217, 2014.

[31] M. Ashton et al., "Ten Things to Look for in an SDN Controller," white paper, 2013. [Online]. Available: www.necam.com/Docs

[32] I. Monga et al., "Operationalization of Software-Defined Networks (SDN) Program Review," NSF White Paper, 12 2013.

[33] P. Berde et al., "ONOS: Towards an Open, Distributed SDN OS," in Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. ACM, 2014, pp. 1–6.

[34] "OPNFV." [Online]. Available: https://www.opnfv.org/software

[35] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-Based Networking with DIFANE," ACM SIGCOMM Computer Communication Review, vol. 41, no. 4, pp. 351–362, 2011.

[36] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in Proceedings of the 2010 internet network management conference on Research on enterprise networking. USENIX Association, 2010, pp. 3–3.

[37] N. Gude, "NOX: Towards an Operating System for Networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 3, pp. 105–110, 2008.

[38] G. Romero, "SNAC," October 2012. [Online]. Available: http://www.valleytalk.org/wp-content/uploads/2013/02/Evaluation_Of_OF_Controllers.pdf

[39] "POX." [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[40] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN Programming with Pyretic," Technical Report of USENIX, 2013.

[41] "Ryu OpenFlow Controller." [Online]. Available: http://osrg.github.io/ryu/

[42] J. H. Cox, S. Donovan, R. J. Clarky, and H. L. Owen, "Ryuretic: A modular framework for Ryu," in Military Communications Conference, MILCOM 2016-2016 IEEE. IEEE, 2016, pp. 1065–1070.

[43] D. Erickson, "The Beacon OpenFlow Controller," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. ACM, 2013, pp. 13–18.

[44] "Floodlight." [Online]. Available: http://www.projectfloodlight.org/floodlight/

[45] "Trema." [Online]. Available: http://trema.github.io/trema/

[46] "Kandoo." [Online]. Available: http://kandoo.github.io/

[47] "OpenContrail." [Online]. Available: http://www.opencontrail.org/opencontrail-architecture-documentation/

[48] "OpenMUL." [Online]. Available: http://www.openmul.org/

[49] "ONF," January. [Online]. Available: https://www.opennetworking.org/

[50] N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008.

[51] D. et al., "Forwarding and Control Element Separation (ForCES) Protocol Specification," March 2010. [Online]. Available: http://sdn.ieee.org/images/files/pdf/Software_Defined_Infrastructure-IEEE-SDN-Initiative.pdf

[52] H. Song, "Protocol-Oblivious Forwarding: Unleash the Power of SDN Through a Future-Proof Forwarding Plane," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. ACM, 2013, pp. 127–132.

[53] N. Feamster, "Coursera, SDN Networks," June 2015. [Online]. Available: https://class.coursera.org/sdn1-001/wiki/

[54] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-Defined Networking (SDN): A Reference Architecture and Open APIs," in ICT Convergence (ICTC), 2012 International Conference on. IEEE, 2012, pp. 360–361.

[55] ONF, "OpenFlow Switch Specification, Version 1.3.0," June 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[56] T. Nolle, "Centralized vs. Decentralized SDN Architecture: Which Works For You?" 2013. [Online]. Available: http://searchsdn.techtarget.com/tip/Centralized-vsdecentralized-SDN-architecture-Which-works-for-you

[57] J. Amann and R. Sommer, "Providing Dynamic Control to Passive Network Security Monitoring," in International Workshop on Recent Advances in Intrusion Detection. Springer, 2015, pp. 133–152.

[58] B. Nunes et al., "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," Communications Surveys & Tutorials, IEEE, vol. 16, no. 3, pp. 1617–1634, 2014.

[59] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," Communications Magazine, IEEE, vol. 51, no. 2, pp. 114–119, 2013.

[60] Y. Yuan, R. Alur, and B. T. Loo, "NetEgg: Programming Network Policies By Examples," in Proceedings of the 13th ACM Workshop on Hot Topics in Networks. ACM, 2014, p. 20.

[61] F. A. Lopes, M. Santos, R. Fidalgo, and S. Fernandes, "A Software Engineering Perspective on SDN Programmability," IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1255–1272, 2016.

[62] Z. Cai, A. L. Cox, and T. E. Ng, "Maestro: A System For Scalable OpenFlow Control," Structure, 2010.

[63] T. Koponen, "Onix: A Distributed Control Platform for Large-scale Production Networks," in OSDI, vol. 10, 2010, pp. 1–6.

[64] "Intent Framework," April 2015. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Intent+Framework

[65] Y. Yuan et al., "Scenario-Based Programming for SDN Policies," 2015.

[66] C.J. Anderson et al., "NetKAT: Semantic Foundations For Networks," ACM SIGPLAN Notices, vol. 49, no. 1, pp. 113–126, 2014.

[67] C. Monsanto et al., "A Compiler and Run-Time System for Network Programming Languages," in ACM SIGPLAN Notices, vol. 47, no. 1. ACM, 2012, pp. 217–230.

[68] X. Jin et al., "CoVisor: A Compositional Hypervisor for Software-Defined Networks," in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), 2015.

[69] D. Lenrow, "Intent: What. Not How." ONF Blog, January 2015. [Online]. Available: https://www.opennetworking.org/?p=1633\&option=com_wordpress\&Itemid=155

[70] S. Donovan and N. Feamster, "Intentional Network Monitoring: Finding the Needle Without Capturing the Haystack," in Proceedings of the 13th ACM Workshop on Hot Topics in Networks, ser. HotNets-XIII. New York, NY, USA: ACM, 2014, pp. 5:1–5:7.

[71] M. Robuck, "ONF Debuts Northbound Interfaces for Intent-Based Networking," September 2015. [Online]. Available: https://www.sdxcentral.com/articles/news/onf-debuts-northbound-interfaces-for-intent-based-networking/2015/09/

[72] B. Chandrasekaran, B. Tschaen, and T. Benson, "Isolating and Tolerating SDN Application Failures with LegoSDN," in Proceedings of the Symposium on SDN Research. ACM, 2016, p. 7.

[73] "HP SDN Dev Center App Store." [Online]. Available: http://www8.hp.com/us/en/networking/sdn/devcenter-index.html

[74] D. Hock et al., "POCO-PLC: Enabling Dynamic Pareto-Optimal Resilient Controller Placement in SDN Networks," in Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on. IEEE, 2014, pp. 115–116.

[75] "POCO-toolset." [Online]. Available: http://www3.informatik.uni-wuerzburg.de/poco

[76] P. Lin, J. Bi, and Y. Wang, "East-West Bridge for SDN Network Peering," in Frontiers in Internet Technologies. Springer, 2013, pp. 170–181.

[77] J. Cox, R. Clark, and H. Owen, "Security Policy Transition Framework for Software Defined Networks," in The First International Workshop on Security in NFV-SDN, 2016. SNS 2016. IEEE. IEEE, 2016.

[78] L. Schiff and S. Schmid, "Study The Past If You Would Define The Future: Implementing Secure Multi-Party SDN Updates," in Software Science, Technology and Engineering (SWSTE), 2016 IEEE International Conference on. IEEE, 2016, pp. 111–116.

[79] A. Dixit et al., "Towards An Elastic Distributed SDN Controller," in ACM SIGCOMM Computer Communication Review, vol. 43, no. 4. ACM, 2013, pp. 7–12.

[80] P. Bosshart et al., "P4: Programming Protocol-Independent Packet Processors," ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 87–95, 2014.

[81] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs From Malicious Administrators," in Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. ACM, 2014, pp. 103–108.

[82] S. Shenker, "Stanford Seminar - Software-Defined Networking at the Crossroads," 2013. [Online]. Available: https://www.youtube.com/watch?v=WabdXYzCAOU

[83] M. Shahbaz et al., "PISCES: A Programmable, Protocol-Independent Software Switch," in Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 525–538.

[84] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks," in Proc. NSDI, 2014.

[85] "Switch Light." [Online]. Available: http://www.bigswitch.com/sites/default/files/sdnresources/switchlightdatasheet.pdf

[86] P Bosshart et al., "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN," in ACM SIGCOMM Computer Communication Review, vol. 43, no. 4. ACM, 2013, pp. 99–110.

[87] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite Cacheflow in Software-Defined Networks," in Proceedings of the Third workshop on Hot Topics in Software Defined Networking. ACM, 2014, pp. 175–180.

[88] P. L. Consortium. P4. [Online]. Available: www.p4.org

[89] N. F. M. Shahbaz, "NetASM-Python," June 2015. [Online]. Available: https://github.com/NetASM/NetASM-python/wiki

[90] B. Pfaff et al., "The Design and Implementation of Open vSwitch," in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), 2015, pp. 117–130.

[91] J. Sherry, S. Ratnasamy, and J. S. At, "A Survey of Enterprise Middlebox Deployments," 2012.

[92] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: Building Robust Firewalls for Software-Defined Networks," in Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. ACM, 2014, pp. 97–102.

[93] J. Sherry et al., "Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service," SIGCOMM Comput. Commun. Rev., vol. 42, no. 4, pp. 13–24, Aug. 2012.

[94] M. Chiosi et al., "Network Functions Virtualisation Introductory White Paper," in SDN and OpenFlow World Congress, 2012.

[95] V. Sekar et al., "Design And Implementation Of A Consolidated Middlebox Architecture," in Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 2012, pp. 323–336.

[96] Z. Qazi, C.-C. Tu, R. Miao, L. Chiang, V. Sekar, and M. Yu, "Practical and Incremental Convergence Between SDN and Middleboxes," Open Network Summit, Santa Clara, CA, 2013.

[97] Z. Qazi et al., "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in ACM SIGCOMM Computer Communication Review, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[98] Fayazbakhsh et al., "Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions Using Flowtags," in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), 2014, pp. 543–546.

[99] M. Casado et al., "Ethane: Taking Control of the Enterprise," in ACM SIGCOMM Computer Communication Review, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[100] ETSI, "Draft ETSI GS NFV-SEC 001 V0. 2.1 (2014-06)," 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/001/01.01.01_60/gs_NFV-SEC001v010101p.pdf

[101] J. Carapinha and J. Jiménez, "Network Virtualization: A View from the Bottom," in Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures. ACM, 2009, pp. 73–80.

[102] "OpenVswitch." [Online]. Available: http://openvswitch.org

[103] R. Sherwood et al., "Flowvisor: A Network Virtualization Layer," OpenFlow Switch Consortium, Tech. Rep, 2009.

[104] R. Jain and S. Paul, "Network Virtualization and Software Defined Networking for Cloud Computing: A Survey," Communications Magazine, IEEE, vol. 51, no. 11, pp. 24–31, 2013.

[105] R. Sherwood 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.

[106] D. Drutskoy, E. Keller, and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," IEEE Internet Computing, vol. 17, no. 2, pp. 20–27, 2013.

[107] Z. Bozakov and P. Papadimitriou, "Autoslice: Automated and Scalable Slicing for Software-Defined Networks," in Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop. ACM, 2012, pp. 3–4.

[108] S. Barkai et al., "Software Defined Flow-Mapping for Scaling Virtualized Network Functions," in Proceedings of the Second ACM SIGCOMM workshop on Hot Topics in Software Defined Networking. ACM, 2013, pp. 149–150.

[109] L. Peterson et al., "Central Office Re-Architected as a Data Center," IEEE Communications Magazine, vol. 54, no. 10, pp. 96–101, 2016.

[110] M. Chiosi et al., "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action," in SDN and OpenFlow World Congress, 2012, pp. 22–24.

[111] J. G. Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," IEEE Transactions on Network and Service Management, vol. 13, no. 3, pp. 518–532, 2016.

[112] A. Al-Shabibi and L. Peterson, "CORD: Central Office Re-architected as a Datacenter," OpenStack Summit, 2015.

[113] P. Pate, "NFV and SDN: WhatâĂŹs the Difference?" SdxCental, March 2013. [Online]. Available: https://www.sdxcentral.com/articles/contributed/nfv-and-sdn-whats-the-difference/2013/03/

[114] Y. Li and M. Chen, "Software-Defined Network Function Virtualization: A Survey," IEEE Access, vol. 3, pp. 2542–2553, 2015.

[115] T. Xing, Z. Xiong, D. Huang, and D. Medhi, "SDNIPS: Enabling Software-Defined Networking Based Intrusion Prevention System in Clouds," in Network and Service Management (CNSM), 2014 10th International Conference on. IEEE, 2014, pp. 308–311.

[116] P. Boyle and J. Cox, "Main Communications Facility Start Up," Army Communicator, vol. 39, p. 61, June 2014.

[117] SDxCentral, "SDN & NFV Use Cases Defined," 2015. [Online]. Available: https://www.sdxcentral.com/sdn-nfv-use-cases/

[118] R. Spillane et al., "Exo-Clones: Better Container Runtime Image Management Across the Clouds," in 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16), 2016.

[119] A. Boubendir, E. Bertin, and N. Simoni, "On-Demand Dynamic Network Service Deployment Over NASS Architecture," in Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP. IEEE, 2016, pp. 1023–1024.

[120] J. Evens, "A Comparison of Containers and Virtual Machines for Use with NFV," MS Thesis. tUL, 2015.

[121] A. Bremler-Barr, Y. Harchol, and D. Hay, "OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions," in Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference. ACM, 2016, pp. 511–524.

[122] B. Anwer et al., "Programming Slick Network Functions," in Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. ACM, 2015, p. 14.

[123] "ATIS." [Online]. Available: https://www.atis.org/NFV/index.asp

[124] E. Kohler et al., "The Click Modular Router," ACM Transactions on Computer Systems (TOCS), vol. 18, no. 3, pp. 263–297, 2000.

[125] J. Martins, "ClickOS and the Art of Network Function Virtualization," in Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, 2014, pp. 459–473.
[126] CloudBand. [Online]. Available: http://resources.alcatel-lucent.com/asset/200060
[127] Zoom. [Online]. Available: https://www.tmforum.org/zoom/
[128] Planet Orchestrate. [Online]. Available: http://www.cyaninc.com/products/blue-planet-sdn-platform
[129] D. Kushner, "The Real Story of Stuxnet," IEEE Spectrum, vol. 50, no. 3, pp. 48–53, 2013.
[130] D. Kreutz, F. Ramos, and P. Verissimo, "Towards Secure and Dependable Software-Defined Networks," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. ACM, 2013, pp. 55–60.
[131] E. Auchard, "Cisco Router Break-Ins Bypass Cyber Defenses," http://www.reuters.com/article/2015/09/15/us-cybersecurity-routers-cisco-systems-idUSKCN0RF0N420150915?feedType=RSS\&feedName=businessNews, September 2015.
[132] R. Kloti, V. Kotronis, and P. Smith, "OpenFlow: A Security Analysis," in Network Protocols (ICNP), 2013 21st IEEE International Conference on. IEEE, 2013, pp. 1–6.
[133] S. Shin et al., "Rosemary: A Robust, Secure, and High-Performance Network Operating System," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014, pp. 78–89.
[134] P. Porras et al., "A Security Enforcement Kernel for OpenFlow Networks," in Proceedings of the First Workshop on Hot Topics in Software Defined Networking. ACM, 2012, pp. 121–126.
[135] K. Benton, L. J. Camp, and C. Small, "Openflow Vulnerability Assessment," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networkingg. ACM, 2013, pp. 151–152.
[136] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures." NDSS, 2015.
[137] U. Toseef, A. Zaalouk, T. Rothe, M. Broadbent, and K. Pentikousis, "C-BAS: Certificate-based AAA for SDN Experimental Facilities," in Software Defined Networks (EWSDN), 2014 Third European Workshop on. IEEE, 2014, pp. 91–96.
[138] S. Betge-Brezetz, G. B. Kamga, and M. Tazi, "Trust Support for SDN Controllers and Virtualized Network Applications," in Network Softwarization (NetSoft), 2015 1st IEEE Conference on, April 2015, pp. 1–5.
[139] A. Kamisiński and C. Fung, "Flowmon: Detecting Malicious Switches in Software-Defined Networks," in Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense. ACM, 2015, pp. 39–45.
[140] J. H. Cox, R. J. Clark, and H. L. Owen, "Leveraging SDN to Improve the Security of DHCP," in SDNNFVSEC, 2016, pp. 1–4.
[141] ——, "Leveraging SDN for ARP security," in SoutheastCon, 2016. IEEE, 2016, pp. 1–8.
[142] ——, "Leveraging SDN and WebRTC for Rogue Access Point Security," IEEE Transactions on Network and Service Management, 2017.
[143] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking," in Proceedings of the First Workshop on Hot topics in Software Defined Networks. ACM, 2012, pp. 127–132.
[144] J. Zheng and A. S. Namin, "The Impact of Address Changes and Host Diversity on the Effectiveness of Moving Target Defense Strategy," in Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual, vol. 2. IEEE, 2016, pp. 553–558.
[145] "HP's SDN App Store." [Online]. Available: https://goo.gl/2vtdHH
[146] J. Qadir and O. Hasan, "Applying Formal Methods to Networking: Theory, Techniques, and Applications," IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 256–291, 2015.
[147] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller Fault-Tolerance in Software-Defined Networking," in Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. ACM, 2015, p. 4.
[148] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," Journal of the ACM (JACM), vol. 27, no. 2, pp. 228–234, 1980.
[149] C. Scott et al., "Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences," ACM SIGCOMM Computer Communication Review, vol. 44, no. 4, pp. 395–406, 2015.
[150] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On Scalability of Software-Defined Networking," Communications Magazine, IEEE, vol. 51, no. 2, pp. 136–141, 2013.
[151] H. Xie et al., "Use Cases for ALTO with Software Defined Networks," Working Draft, IETF Secretariat, Internet-Draft draft-xie-alto-sdn-extension-use-cases-01. txt, 2012.
[152] R. Sherwood and K. Yap, "Cbench Controller Benchmarker," Nov 2014. [Online]. Available: https://github.com/andi-bigswitch/oflops/tree/master/cbench
[153] M. Jarschel et al., "OFCProbe: A Platform-Independent Tool for OpenFlow Controller Analysis," in Communications and Electronics (ICCE), 2014 IEEE Fifth International Conference on. IEEE, 2014, pp. 182–187.
[154] "Mininet." [Online]. Available: www.mininet.org
[155] G. Gee, "MiniEdit 2.2.0.1," Tech and Trains. [Online]. Available: https://techandtrains.com/category/miniedit/
[156] J. Cheng et al., "Towards a Detailed OpenFlow Emulator," in Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific, Aug 2015, pp. 127–132.
[157] P. Danielis et al., "Emulation of SDN-Supported Automation Networks," in 2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA), Sept 2015, pp. 1–8.
[158] V. Antonenko and R. Smelyanskiy, "Global Network Modeling Based on Mininet Approach." in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 145–146.
[159] M. Chan et al., "OpenNet: A Simulator for Software-Defined Wireless Local Area Network," in Wireless Communications and Networking Conference (WCNC), 2014 IEEE. IEEE, 2014, pp. 3332–3336.
[160] "Ns-3." [Online]. Available: https://www.nsnam.org/overview/what-is-ns-3/
[161] R. Fontes et al., "Mininet-WiFi: Emulating Software-Defined Wireless Networks," in Network and Service Management (CNSM), 2015 11th International Conference on. IEEE, 2015, pp. 384–389.
[162] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "EstiNet OpenFlow Network Simulator and Emulator," Communications Magazine, IEEE, vol. 51, no. 9, pp. 110–117, 2013.
[163] S. Y. Wang, "Comparison of SDN OpenFlow Network Simulator and Emulators: EstiNet vs. Mininet," in 2014 IEEE Symposium on Computers and Communications (ISCC), June 2014, pp. 1–6.
[164] J. Sommers et al., "Efficient Network-Wide Flow Record Generation," in INFOCOM, 2011 Proceedings IEEE, April 2011, pp. 2363–2371.
[165] M. Gupta, J. Sommers, and P. Barford, "Fast, Accurate Simulation for SDN Prototyping," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 31–36.
[166] A.R. Roy et al., "Design and Management of DOT: A Distributed OpenFlow Testbed," in Network Operations and Management Symposium (NOMS), 2014 IEEE, May 2014, pp. 1–9.
[167] B. Hurd, "OpenFlow Switch Support – Model Library," Ns-3, 2011. [Online]. Available: https://www.nsnam.org/docs/models/html/openflow-switch.html
[168] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "OFSwitch13: Enhancing Ns-3 with OpenFlow 1.3 Support," in Proceedings of the Workshop on Ns-3. ACM, 2016, pp. 33–40.
[169] E. L. Fernandes and C. E. Rothenberg, "OpenFlow 1.3 Software Switch," Salao de Ferramentas do XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos SBRC, pgs, pp. 1021–1028, 2014.
[170] J. Ivey et al., "Comparing a Scalable SDN Simulation Framework Built on Ns-3 and DCE with Existing SDN Simulators and Emulators," in Proceedings of the 2016 Annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation, ser. SIGSIM-PADS '16. ACM, 2016, pp. 153–164.
[171] B. White et al., "An Integrated Experimental Environment for Distributed Systems and Networks," in Proc. of the Fifth Symposium on Operating Systems Design and Implementation. Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.
[172] A. Bavier et al., "Operating System Support for Planetary-scale Network Services," in Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1, ser. NSDI'04. Berkeley, CA, USA: USENIX Association, 2004, p. 19.
[173] M. Berman et al., "GENI: A Federated Testbed for Innovative Network Experiments," Computer Networks, vol. 61, no. 0, pp. 5 – 23, 2014, special issue on Future Internet Testbeds âĂŞ Part I.

[174] A. C. Risdianto, T. Na, and J. Kim, "Running Lifecycle Experiments Over SDN-Enabled OF@TEIN Testbed," in 2014 IEEE Fifth International Conference on Communications and Electronics (ICCE), July 2014, pp. 194–198.

[175] M. SuÃśÃl' et al., "Design and Implementation of the {OFELIA} {FP7} Facility: The European OpenFlow Testbed," Computer Networks, vol. 61, pp. 132 – 150, 2014, special issue on Future Internet Testbeds âĂŞ Part I.

[176] Y. Kanaumi, "RISE: A Wide-Area Hybrid OpenFlow Network Testbed," IEICE Trans. on Commun., vol. 96, no. 1, pp. 108–118, 2013.

[177] W. C. Chung et al., "Taiwan UniCloud: A Cloud Testbed with Collaborative Cloud Services," in 2014 IEEE International Conference on Cloud Engineering, March 2014, pp. 107–116.

[178] J. Kang et al., Software-Defined Infrastructure and the SAVI Testbed. Cham: Springer International Publishing, 2014, pp. 3–13.

[179] T. Huang et al., "A Survey on Large-scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges," IEEE Communications Surveys Tutorials, vol. PP, no. 99, pp. 1–1, 2016.

[180] D. Levin, M. Canini, S. Schmid, A. Feldmann et al., "Toward Transitional SDN Deployment in Enterprise Networks," Proceedings of the Open Networking Summit (ONS), 2013.

[181] D. Levin, M. Canini, S. Schmid, F. Schaffert, A. Feldmann et al., "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks," in USENIX ATC, 2014.

[182] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and Research Challenges of Hybrid Software Defined Networks," ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 70–75, 2014.

[183] C. E. Rothenberg, A. Vidal, M. R. Salvador, C. Correa, S. Lucena, F. Farias, E. Cerqueira, and A. Abelem, "Hybrid Networking Towards a Software Defined Era," in Network Innovation through OpenFlow and SDN: Principles and Design book, Taylor & Francis LLC, CRC Press., 2014.

[184] "Google Inc. Inter-data center WAN with centralized TE using SDN and OpenFlow," 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-googlesdn.pdf

[185] N. Feamster, "Interview with Bryan Larish," June 2015. [Online]. Available: https://www.youtube.com/watch?v=gdnyIiQR0Eg

[186] R. Clark, "The Software Defined Exchange - GEC 23," 2015. [Online]. Available: http://groups.geni.net/geni/attachment/wiki/GEC23Agenda/SDX/GT-SDX-GEC23.pdf

[187] J. Mambretti, J.-J. Chen, and F. Yeh, "Software-Defined Network Exchanges (SDXs) and Infrastructure (SDI): Emerging innovations in SDN and SDI Interdomain Multi-Layer Services and Capabilities," in Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 First International. IEEE, 2014, pp. 1–6.

[188] J. Chung et al., "AtlanticWave-SDX: An International SDX to Support Science Data Applications," in High Performance Computing, Networking, Storage, and Analysis, SC15: International Conference for. IEEE, 2015.

[189] "IRNC: RXP: Starlight SDX A Software Defined Networking Exchange for Global Science Researchers." [Online]. Available: http://internationalnetworking.iu.edu/files/ppt/ppt/iCAIR%20NSI-SDN-SDX%20Presentation%20April%202014.pdf

[190] J. Stringer et al., "Cardigan: SDN Distributed Routing Fabric Going Live at an Internet Exchange," in Computers and Communication (ISCC), 2014 IEEE Symposium on. IEEE, 2014, pp. 1–7.

[191] I. Monga, "Software Defined Exchanges: The new SDN?" 2014. [Online]. Available: http://www.ieice.org/~nv/nvs2014/nvs2014-05-monga.pdf

[192] R. Ricci and N. Feamster, Eds., Report of the NSF Workshop on Software Defined Infrastructures and Software Defined Exchanges, Washington, DC, Feb. 2016.

[193] J. Chung, R. Clark, and H. Owen, "SDX Architectures: A Qualitative Analysis," in IEEE SoutheastCon 2016. IEEE, 2016, pp. 1–8.

[194] B. Ager et al., "Anatomy of a Large European IXP," in Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication. ACM, 2012, pp. 163–174.

[195] J. Ibarra et al., "Benefits Brought by the Use of OpenFlow/SDN in the AmLight Intercontinental Research and Education Network."

[196] NSF, "International Research Network Connections (IRNC)," http://www.nsf.gov/pubs/2014/nsf14554/nsf14554.htm.

[197] X. Yang and T. Lehman, "Software Defined Services: Exchange Points (SDX) and ScienceDMZs (SD-SDMZ)." Presented at Internet2 Technology Exchange, 2016, 2016.

[198] J. Mambretti, J. Chen, and F. Yeh, "Software-Defined Network Exchanges (SDXS): Architecture, Services, Capabilities, and Foundation Technologies," in Teletraffic Congress (ITC), 2014 26th International. IEEE, 2014, pp. 1–6.

[199] I. Baldine, "An Advanced International Distributed Programmable Environment for Experimental Network Research:âĂIJSlice Around the WorldâĂİ Demonstration." Global LambdaGrid Workshop, 2016.

[200] A. Gupta et al., "An Industrial-Scale Software Defined Internet Exchange Point," in 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). USENIX Association, 2016, pp. 1–14.

[201] P. Lin, J. Bi, S. Wolff, Y. Wang, A. Xu, Z. Chen, H. Hu, and Y. Lin, "A West-East Bridge based SDN Inter-Domain Testbed," Communications Magazine, IEEE, vol. 53, no. 2, pp. 190–197, 2015.

[202] K. Pentikousis, "Software Defined Infrastructure: The FELIX Architecture Blueprint and Implementation Experience," 2015. [Online]. Available: http://sdn.ieee.org/images/files/pdf/Software_Defined_Infrastructure-IEEE-SDN-Initiative.pdf

[203] S. Donovan, J. Chung, M. Sanders, and R. Clark, "MetroSDX: A Resilient Edge for the Smart Community," in SmartEdge 2017. IEEE, 2017.

[204] J. Griffioen, T. Wolf, and K. L. Calvert, "A Coin-Operated Software-Defined Exchange," in Computer Communication and Networks (ICCCN), 2016 25th International Conference on. IEEE, 2016, pp. 1–8.

[205] R. di Lallo, "On the Practical Applicability of SDN Research," in Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP. IEEE, 2016, pp. 1–9.

[206] T. Hinrichs et al., "Expressing and Enforcing Flow-Based Network Security Policies," University of Chicago, Tech. Rep, vol. 9, 2008.

[207] R. Soulé et al., "Managing the Network with Merlin," in Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks. ACM, 2013, p. 24.

[208] A. Khurshid et al., "Veriflow: Verifying Network-Wide Invariants in Real Time," in Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013, pp. 15–27.

[209] M. Reitblatt et al., "Fattire: Declarative Fault Tolerance for Software-Defined Networks," in Proceedings of the second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. ACM, 2013, pp. 109–114.

[210] P. Porras et al., "A Security Enforcement Kernel for OpenFlow Networks," in Proceedings of the First Workshop on Hot Topics in Software Defined Networking. ACM, 2012, pp. 121–126.

[211] P. Kazemian et al., "Real Time Network Policy Checking Using Header Space Analysis," in Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013, pp. 99–111.

[212] F. Maldonado et al., "Checking Multi-domain Policies in SDN," International Journal of Computers Communications & Control, vol. 11, no. 3, pp. 428–440, 2016.

[213] 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.

[214] M. Palattella et al., "Internet of Things in the 5G Era: Enablers, Architecture, and Business Models," IEEE Journal on Selected Areas in Communications, vol. 34, no. 3, pp. 510–527, 2016.

[215] S. Jontz, "Security Concerns Rising in the Age of IoT," Signal, pp. 25–28, March 2017.

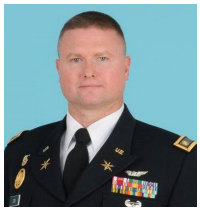
[216] K. Hong et al., "Mobile Fog: A Programming Model for Large-Scale Applications on the Internet of Things," in Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing. ACM, 2013, pp. 15–20.

[217] A. M. Alberti and D. Singh, "Internet of Things: Perspectives, Challenges and Opportunities," in International Workshop on Telecommunications (IWT 2013), 2013.

[218] C. Kruger, A. Abu-Mahfouz, and G. Hancke, "Rapid Prototyping of a Wireless Sensor Network Gateway for the Internet of Things Using Off-The-Shelf Components," in Industrial Technology (ICIT), 2015 IEEE International Conference on. IEEE, 2015, pp. 1926–1931.

[219] A. Mahmud, R. Rahmani, and T. Kanter, "Deployment of Flow-Sensors in Internet of Things' Virtualization via OpenFlow," in Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on. IEEE, 2012, pp. 195–200.

[220] D. Wu et al., "UbiFlow: Mobility Management in Urban-Scale Software Defined IoT," in Computer Communications (INFOCOM), 2015 IEEE Conference on. IEEE, 2015, pp. 208–216.
[221] I. Ku et al., "Towards Software-Defined VANET: Architecture and Services," in Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual Mediterranean. IEEE, 2014, pp. 103–110.
[222] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards sdn-defined programmable byod (bring your own device) security," Network and Distributed System Security Symposium (NDSS 2016), Feb 2016.
[223] D. Syrivelis et al., "Pursuing a Software Defined Information-Centric Network," in Software Defined Networking (EWSDN), 2012 European Workshop on. IEEE, 2012, pp. 103–108.
[224] S. Salsano et al., "Information Centric Networking Over SDN and OpenFlow: Architectural Aspects and Experiments on the OFELIA Testbed," Computer Networks, vol. 57, no. 16, pp. 3207–3221, 2013.
[225] J. Ren, K. Lu, S. Wang, X. Wang, S. Xu, L. Li, and S. Liu, "VICN: A Versatile Deployment Framework for Information-Centric Networks," Network, IEEE, vol. 28, no. 3, pp. 26–34, 2014.
[226] C. Liang, F. R. Yu, and X. Zhang, "Information-centric network function virtualization over 5g mobile wireless networks," IEEE Network, vol. 29, no. 3, pp. 68–74, 2015.
[227] Y. Zhang and Y. Wang, "Sdn based icn architecture for the future integration network," in Communications and Information Technologies (ISCIT), 2016 16th International Symposium on. IEEE, 2016, pp. 474–478.
[228] J. F. Monserrat, G. Mange, V. Braun, H. Tullberg, G. Zimmermann, and Ö. Bulakci, "Metis research advances towards the 5g mobile and wireless system definition," EURASIP Journal on Wireless Communications and Networking, vol. 2015, no. 1, p. 1, 2015.
[229] D. Sahinel et al., "Beyond 5G Vision for IOLITE Community," IEEE Communications Magazine, vol. 55, no. 1, pp. 41–47, 2017.
[230] N. Alliance, "5g white paper-executive version," White Paper, December, 2014.
[231] E. Hossain and M. Hasan, "5g cellular: key enabling technologies and research challenges," IEEE Instrumentation & Measurement Magazine, vol. 18, no. 3, pp. 11–21, 2015.
[232] I. F. Akyildiz, P. Wang, and S.-C. Lin, "Softair: A software defined networking architecture for 5g wireless systems," Computer Networks, vol. 85, pp. 1–18, 2015.
[233] N. Bhushan et al., "Network densification: the dominant theme for wireless evolution into 5g," IEEE Communications Magazine, vol. 52, no. 2, pp. 82–89, 2014.
[234] A. M. Akhtar, X. Wang, and L. Hanzo, "Synergistic Spectrum Sharing in 5G HetNets: A harmonized SDN-enabled approach."
[235] X. Duan and X. Wang, "Authentication handover and privacy protection in 5g hetnets using software-defined networking," IEEE Communications Magazine, vol. 53, no. 4, pp. 28–35, 2015.
[236] C. Chaudet and Y. Haddad, "Wireless Software Defined Networks: Challenges and Opportunities," in Microwaves, Communications, Antennas and Electronics Systems (COMCAS), 2013 IEEE International Conference on. IEEE, 2013, pp. 1–5.
[237] A. Hakiri et al., "Software-Defined Networking: Challenges and Research Opportunities for Future Internet," Computer Networks, vol. 75, pp. 453–471, 2014.
[238] "MiniNext." [Online]. Available: https://github.com/sdn-ixp/sdx-pyretic/wiki/Installation-Instructions
[239] A. Gudipati et al., "SoftRAN: Software defined radio access network," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. ACM, 2013, pp. 25–30.
[240] Y. Cai et al., "Software-Defined Device-to-Device (D2D) Communications in Virtual Wireless Networks With Imperfect Network State Information (NSI)," IEEE Transactions on Vehicular Technology, vol. 65, no. 9, pp. 7349–7360, 2016.



Jacob H. Cox Jr. received his BSEE from Clemson University, SC in 2002, his MSECE from Duke University, NC in 2010, and his Ph.D. in ECE from Georgia Institute of Technology in 2017. As an Army officer, Jacob has served as a Cyber officer, a Telecom engineer, and Signal officer. His recent assignments include Company Command at Fort Gordon, Georgia; Assistant Professor at the USMA, West Point, NY; and Enterprise Operations Chief at the South West Asia Cyber Center in Kuwait. In 2017, he joined Soar Technology, Inc. as a Research Scientist. His research interests include software-defined networking, network control platforms, and network security.

Joaquin Chung received both his B.S. in Electrics and Communications Engineering (2007) and his M.S. in Communication Systems Engineering with emphasis in Data Networks (2013) from University of Panama, Panama. He is a Fulbright scholar and currently pursuing his Ph.D. in Electrical and Computer Engineering under the supervision of Dr. Henry Owen and Dr. Russell Clark at Georgia Institute of Technology, GA. His research interest include software-defined networking and network security.

Sean Donovan received his B.S. in Computer Science from Worcester Polytechnic Institute in 2006, and M.S. in Computer Science from Georgia Institute of Technology in 2016. Sean currently works as a research scientist for the Georgia Tech RNOC focusing on software-defined networking (SDN), software-defined exchanges (SDXes), and information and communications technology (ICT) in smart cities. His previous experience includes software engineering for networking and network protocols.

Jared Ivey received his B.S. in Biomedical Engineering from the Georgia Institute of Technology in May 2009, his M.S.E. in Software Engineering from Mercer University in May 2012, and his Ph.D. in Electrical and Computer Engineering in 2017 from the Georgia Institute of Technology. His research interests focus on providing scalable and reliable solutions for network simulation in SDN. He is currently employed as a technical lead engineer for software development and maintenance of the Joint STARS platform at Robins AFB, GA.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2762291, IEEE Access



Russell Clark received the B.S. in Mathematics and Computer Science from Vanderbilt University in 1987. He received the M.S. and Ph.D. degrees in Information and Computer Science from Georgia Institute of Technology in 1992 and 1995. For the years 1997-2000 he was a Senior Scientist with Empire Technologies, a network management software company. He is currently a senior research scientist at Georgia Tech's School of Computer Science where he engages hundreds of students each semester in mobile development, networking and the Internet of Things.

George Riley received his Ph.D. from the Georgia Tech College of Computing in August 2001, and joined the faculty of ECE at that time. Dr. Riley received a MSCS from Florida Tech in 1996, and a BSEE from University of Alabama in 1972. Prior to enrolling at Tech in 1996, Dr. Riley was president and CEO of Infoware, Inc. of Cocoa Beach Florida. From 1987 to 1996 Infoware provided software and system design services to the United States Air Force at Patrick Air Force Base, Florida. During that time, Infoware designed, implemented, and deployed numerous systems in support of the missile launch activities at Cape Canaveral Air Force Station, including a communications front-end processor for real-time data gathering and a real-time distributed flight safety display system. Concurrently, from 1984 to 2000, Dr. Riley was also vice-president and co-owner of CAM Systems Inc. of Atlanta Georgia. CAM systems developed, under Dr. Riley's direction, a suite of PC based software tools for residential property management.

Henry Owen received his BSEE, MSEE, and Ph.D. in Electrical Engineering from the Georgia Institute of Technology in 1980, 1983, and 1989 respectively. He joined the research faculty of the Georgia Tech Research Institute in 1980. After completing his Ph.D. in 1989, he joined the academic faculty. In 1990, he took a one-year leave of absence to conduct on site research for Alcatel in Stuttgart, Germany. In 2000, he was involved in an Internet Security start up on a half-time basis. In 2015, he was on Sabbatical at TU Berlin in the Internet Networking Architectures research group of Dr. Anja Feldmann.