acmqueue

The Road to SDN

An intellectual history of programmable networks

Nick Feamster, Georgia Institute of Technology
Jennifer Rexford, Princeton University
Ellen Zegura, Georgia Institute of Technology

Designing and managing networks has become more innovative over the past few years with the aid of SDN (software-defined networking). This technology seems to have appeared suddenly, but it is actually part of a long history of trying to make computer networks more programmable.

Computer networks are complex and difficult to manage. They involve many kinds of equipment, from routers and switches to middleboxes such as firewalls, network address translators, server load balancers, and intrusion-detection systems. Routers and switches run complex, distributed control software that is typically closed and proprietary. The software implements network protocols that undergo years of standardization and interoperability testing. Network administrators typically configure individual network devices using configuration interfaces that vary between vendors—and even between different products from the same vendor. Although some network-management tools offer a central vantage point for configuring the network, these systems still operate at the level of individual protocols, mechanisms, and configuration interfaces. This mode of operation has slowed innovation, increased complexity, and inflated both the capital and the operational costs of running a network.

SDN is changing the way networks are designed and managed. It has two defining characteristics. First, SDN separates the *control plane* (which decides how to handle the traffic) from the *data plane* (which forwards traffic according to decisions that the control plane makes). Second, SDN consolidates the control plane, so that a single software control program controls *multiple* data-plane elements. The SDN control plane exercises direct control over the state in the network's data-plane elements (i.e., routers, switches, and other middleboxes) via a well-defined API. OpenFlow[51] is a prominent example of such an API. An OpenFlow switch has one or more tables of packet-handling rules. Each rule matches a subset of traffic and performs certain actions on the traffic that matches a rule; actions include dropping, forwarding, or flooding. Depending on the rules installed by a controller application, an OpenFlow switch can behave as a router, switch, firewall, network address translator, or something in between.

SDN has gained significant traction in the industry. Many commercial switches support the OpenFlow API. HP, NEC, and Pronto were among the first vendors to support OpenFlow; this list has since expanded dramatically. Many different controller platforms have emerged.[23, 30, 37, 46, 55, 63, 80] Programmers have used these platforms to create many applications, such as dynamic access control,[16,53] server load balancing,[39,81] network virtualization,[54,67] energy-efficient networking,[42] and seamless virtual-machine migration and user mobility.[24] Early commercial successes, such as Google's wide-area traffic-management system[44] and Nicira's Network Virtualization Platform,[54] have garnered significant industry attention. Many of the world's largest information-technology

companies (e.g., cloud providers, carriers, equipment vendors, and financial services firms) have joined SDN industry consortia such as the Open Networking Foundation[57] and the Open Daylight initiative.[56]

Although the excitement about SDN has become more palpable fairly recently, many of the ideas underlying the technology have evolved over the past 20 years (or more). In some ways, SDN revisits ideas from early telephony networks, which used a clear separation of control and data planes to simplify network management and the deployment of new services. Yet, open interfaces such as OpenFlow enable more innovation in controller platforms and applications than was possible on closed networks designed for a narrow range of telephony services. In other ways, SDN resembles past research on active networking, which articulated a vision for programmable networks, albeit with an emphasis on programmable data planes. SDN also relates to previous work on separating the control and data planes in computer networks.

This article presents an intellectual history of programmable networks culminating in present-day SDN. It looks at the evolution of key ideas, the application "pulls" and technology "pushes" of the day, and lessons that can help guide the next set of SDN innovations. Along the way, it debunks myths and misconceptions about each of the technologies and clarifies the relationship between SDN and related technologies such as network virtualization.

The history of SDN began 20 years ago, just as the Internet was taking off, at a time when the Internet's amazing success exacerbated the challenges of managing and evolving the network infrastructure. The focus here is on innovations in the networking community (whether by researchers, standards bodies, or companies), although these innovations were in some cases catalyzed by progress in other areas, including distributed systems, operating systems, and programming languages. The efforts to create a programmable network infrastructure also clearly relate to the long thread of work on supporting programmable packet processing at high speeds.[5, 21, 38, 45, 49, 71, 73]

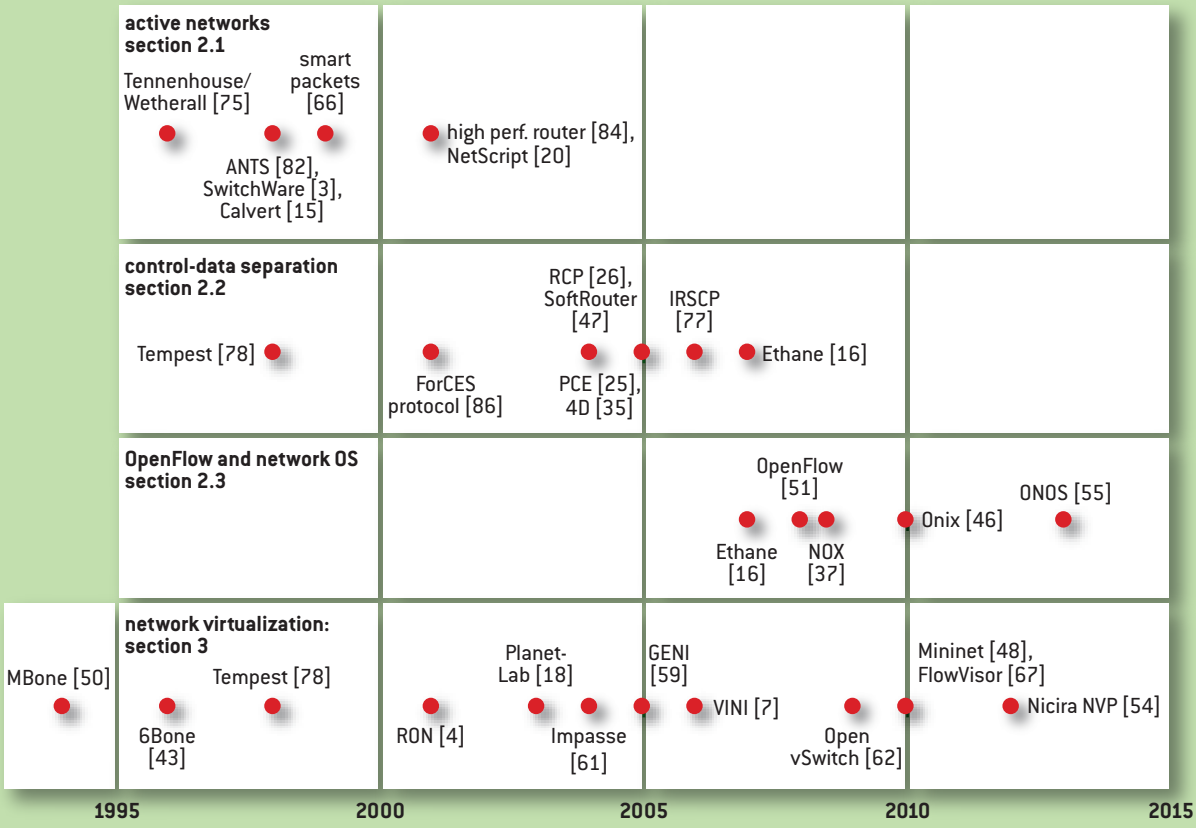
Before beginning this story, we caution the reader that any history is more nuanced than a single storyline might suggest. In particular, much of the work described in this article predates the term *SDN*, coined in an article[36] about the OpenFlow project at Stanford University. The etymology of the term is itself complex, and, although the term was initially used to describe Stanford's OpenFlow project, the definition has since expanded to include a wider array of technologies. (The term has even been co-opted by industry marketing departments to describe unrelated ideas that predated Stanford's SDN project.) Thus, instead of attempting to attribute direct influence between projects, this article highlights the evolution of the ideas that represent the defining characteristics of SDN, regardless of whether they directly influenced subsequent research. Some of these early ideas may not have directly influenced later ones, but the connections between the concepts are noteworthy, and these projects of the past may yet offer new lessons for SDN in the future.

THE ROAD TO SDN

Making computer networks more programmable makes innovation in network management possible and lowers the barrier to deploying new services. This section reviews early work on programmable networks. Figure 1 shows selected developments in programmable networking over the past 20 years and their chronological relationship to advances in network virtualization (one of the first successful SDN use cases).

FIGURE 1

Selected Developments in Programmable Networking Over the Past 20 Years



The history is divided into three stages, each with its own contributions: (1) active networks (from the mid-1990s to the early 2000s), which introduced programmable functions in the network, leading to greater innovation; (2) control- and data-plane separation (from around 2001 to 2007), which developed open interfaces between the control and data planes; and (3) the OpenFlow API and network operating systems (from 2007 to around 2010), which represented the first widespread adoption of an open interface and developed ways to make control- and data-plane separation scalable and practical. Network virtualization (discussed in the next section) played an important role throughout the historical evolution of SDN, substantially predating SDN yet taking root as one of the first significant use cases for SDN.

ACTIVE NETWORKING

The early to mid-1990s saw the Internet take off, with applications and appeal that far outpaced the early applications of file transfer and e-mail for scientists. More diverse applications and greater use by the general public drew researchers who were eager to test and deploy new ideas for improving network services. To do so, researchers designed and tested new network protocols in small lab settings and simulated behavior on larger networks. Then, if motivation and funding continued,

they took their ideas to the IETF (Internet Engineering Task Force) to standardize these protocols, but this was a slow process that ultimately frustrated many researchers.

In response, some networking researchers pursued an alternative approach of opening up network control, roughly based on the analogy of reprogramming a stand-alone PC with relative ease. Conventional networks are not "programmable" in any meaningful sense of the word. *Active networking* represented a radical approach to network control by envisioning a programming interface (or network API) that exposed resources (e.g., processing, storage, and packet queues) on individual network nodes and supported the construction of custom functionality to apply to a subset of packets passing through the node.

This approach was anathema to many in the Internet community who advocated that simplicity in the network core was critical to Internet success. The active networks research program explored radical alternatives to the services provided by the traditional Internet stack via IP or ATM (Asynchronous Transfer Mode), the other dominant networking approach of the early 1990s. In this sense, active networking was the first in a series of clean-slate approaches to network architecture[14] subsequently pursued in programs such as GENI (Global Environment for Network Innovations)[33] and NSF FIND (Future Internet Design)[31] in the United States, and EU FIRE (Future Internet Research and Experimentation Initiative)[32] in the European Union.

The active networking community pursued two programming models:

- The *capsule model*, where the code to execute at the nodes was carried in-band in data packets.[82]
- The *programmable router/switch model*, where the code to execute at the nodes was established by out-of-band mechanisms.[8,68]

The capsule model came to be most closely associated with active networking. In intellectual connection to subsequent efforts, though, both models have a lasting legacy. Capsules envisioned installation of new data-plane functionality across a network, carrying code in data packets (as in earlier work on packet radio[88]) and using caching to improve the efficiency of code distribution. Programmable routers placed decisions about extensibility directly in the hands of the network operator.

Technology push and use pull. The "technology pushes" that encouraged active networking included a reduction in the cost of computing, allowing more processing in the network; advances in programming languages such as Java that offered platform portability and some code execution safety; and virtual machine technology that protected the host machine (in this case the active node) and other processes from misbehaving programs.[70] Some active-networking research projects also capitalized on advances in rapid code compilation and formal methods.

An important catalyst in the active-networking ecosystem was funding agency interest, in particular the Active Networks program created and supported by DARPA (U.S. Defense Advanced Research Projects Agency) from the mid-1990s into the early 2000s. Although not all research into active networks was funded by DARPA, the funding program supported a collection of projects and, perhaps more important, encouraged convergence on a terminology and set of active network components so that projects could contribute to a whole meant to be greater than the sum of the parts.[14] The Active Networks program emphasized demonstrations and project interoperability, with a concomitant level of development effort. The bold and concerted push from a funding agency in the absence of near-term use cases may have also contributed to a degree of community skepticism about active networking that was often healthy but could border on hostility, and it may

have obscured some of the intellectual connections between that work and later efforts to provide network programmability.

The "use pulls" for active networking described in the literature of the time[15,74] are remarkably similar to the examples used to motivate SDN today. The issues of the day included network service provider frustration with the time needed to develop and deploy new network services (so-called network ossification); third-party interest in value-added, fine-grained control to dynamically meet the needs of particular applications or network conditions; and researcher desire for a platform that would support experimentation at scale. Additionally, many early papers on active networking cited the proliferation of middleboxes, including firewalls, proxies, and transcoders, each of which had to be deployed separately and entailed a distinct (often vendor-specific) programming model. Active networking offered a vision of unified control over these middleboxes that could ultimately replace the ad hoc, one-off approaches to managing and controlling these boxes.[74] Interestingly, the early literature foreshadows the current trends in NFV (network functions virtualization),[19] which also aims to provide a unifying control framework for networks that have complex middlebox functions deployed throughout.

Intellectual contributions. Active networks offered intellectual contributions that relate to SDN. Here are three of particular note:

• ***Programmable functions in the network that lower the barrier to innovation.*** Research in active networks pioneered the notion of programmable networks as a way of lowering the barrier to network innovation. The notion that it is difficult to innovate in a production network and pleas for increased programmability were commonly cited in the initial motivation for SDN. Much of the early vision for SDN focused on control-plane programmability, whereas active networks focused more on data-plane programmability. That said, data-plane programmability has continued to develop in parallel with control-plane efforts,[5, 21] and data-plane programmability is again coming to the forefront in the emerging NFV initiative. Recent work on SDN is exploring the evolution of SDN protocols such as OpenFlow to support a wider range of data-plane functions.[11] Also, the concepts of isolation of experimental traffic from normal traffic—which have their roots in active networking—also appear front and center in design documents for OpenFlow[51] and other SDN technologies (e.g., FlowVisor[29]).

• ***Network virtualization and the ability to demultiplex to software programs based on packet headers.*** The need to support experimentation with multiple programming models led to work on network virtualization. Active networking produced an architectural framework that describes the components of such a platform.[13] The key components of this platform are a shared NodeOS (node operating system) that manages shared resources; a set of EEs (execution environments), each of which defines a virtual machine for packet operations; and a set of AAs (active applications) that work within a given EE to provide an end-to-end service. Directing packets to a particular EE depends on fast pattern matching on header fields and demultiplexing to the appropriate EE. Interestingly, this model was carried forward in the PlanetLab[60] architecture, whereby different experiments run in virtual execution environments and packets are demultiplexed into the appropriate execution environment on their packet headers. Demultiplexing packets into different virtual execution environments has also been applied to the design of virtualized programmable hardware data planes.[5]

• ***The vision of a unified architecture for middlebox orchestration.*** Although the vision was never

fully realized in the active-networking research program, early design documents cited the need for unifying the wide range of middlebox functions with a common, safe programming framework. Although this vision may not have directly influenced the more recent work on NFV, various lessons from active-networking research may prove useful as the application of SDN-based control and orchestration of middleboxes moves forward.

Myths and misconceptions. Active networking included the notion that a network API would be available to end users who originate and receive packets, though most in the research community fully recognized that end-user network programmers would be rare.[15] The misconception that packets would necessarily carry Java code written by end users made it possible to dismiss active-network research as too far removed from real networks and inherently unsafe. Active networking was also criticized at the time for its inability to offer practical performance and security. While performance was not a first-order consideration of the active-networking research community (which focused on architecture, programming models, and platforms), some efforts aimed to build high-performance active routers.[84] Similarly, while security was under-addressed in many of the early projects, the secure active network environment (SANE) architecture project[2] was a notable exception.

In search of pragmatism. Although active networks articulated a vision of programmable networks, the technologies did not see widespread deployment. Perhaps one of the biggest stumbling blocks was the lack of an immediately compelling problem or a clear path to deployment. A significant lesson from the active-network research effort was that killer applications for the data plane are hard to conceive. The community proffered various applications that could benefit from in-network processing, including information fusion, caching and content distribution, network management, and application-specific quality of service.[15,74] Unfortunately, although performance benefits could be quantified in the lab, none of these applications demonstrated a sufficiently compelling solution to a pressing need.

Subsequent efforts, described in the next subsection, focused more narrowly on routing and configuration management. In addition to a narrower scope, the next phase of research developed technologies that drew a clear distinction and separation between the functions of the control and data planes. This separation ultimately made it possible to focus on innovations in the control plane, which not only needed a significant overhaul but, because it is commonly implemented in software, presented a lower barrier to innovation than the data plane.

SEPARATING CONTROL AND DATA PLANES

In the early 2000s, increasing traffic volumes and a greater emphasis on network reliability, predictability, and performance led network operators to seek better approaches to certain network management functions, such as control of the paths used to deliver traffic (commonly known as traffic engineering). The means for traffic engineering using conventional routing protocols were primitive at best. Operators' frustration with these approaches was recognized by a small, well-situated community of researchers who either worked for or regularly interacted with backbone network operators. These researchers explored pragmatic, near-term approaches that were either standards-driven or imminently deployable using existing protocols.

Specifically, conventional routers and switches embody a tight integration between the control and data planes. This coupling made various network management tasks, such as debugging

configuration problems and predicting or controlling routing behavior, exceedingly challenging. To address these challenges, various efforts to separate the data and control planes began to emerge.

Technology push and use pull. As the Internet flourished in the 1990s, the link speeds in backbone networks grew rapidly, leading equipment vendors to implement packet-forwarding logic directly in hardware, separate from the control-plane software. In addition, ISPs (Internet service providers) were struggling to manage the increasing size and scope of their networks, as well as the demands for greater reliability and new services (such as virtual private networks). In parallel with these trends, the rapid advances in commodity computing platforms meant that servers often had substantially more memory and processing resources than the control-plane processor of a router deployed just one or two years earlier. These trends catalyzed two innovations:

- An open interface between the control and data planes, such as the ForCES (Forwarding and Control Element Separation)[86] interface standardized by the IETF and the Netlink interface to the kernel-level packet-forwarding functionality in Linux.[65]
- Logically centralized control of the network, as seen in the RCP (Routing Control Platform)[12,26] and SoftRouter[47] architectures, as well as the PCE (Path Computation Element)[25] protocol at the IETF.

These innovations were driven by industry's demands for technologies to manage routing within an ISP network. Some early proposals for separating the data and control planes also came from academic circles, in both ATM[10,30,78] and active networks.[69]

Compared with earlier research on active networking, these projects focused on pressing problems in network management, with an emphasis on innovation by and for network administrators (rather than end users and researchers); programmability in the control plane (rather than the data plane); and network-wide visibility and control (rather than device-level configuration).

Network management applications included selecting better network paths based on the current traffic load, minimizing transient disruptions during planned routing changes, giving customer networks more control over the flow of traffic, and redirecting or dropping suspected attack traffic. Several control applications ran in operational ISP networks using legacy routers, including the IRSCP (Intelligent Route Service Control Point) deployed to offer value-added services for virtual private network customers in AT&T's tier-1 backbone network.[77] Although much of the work during this time focused on managing routing within a single ISP, some work[25,26] also proposed ways to enable flexible route control across multiple administrative domains.

Moving control functionality off of network equipment and into separate servers made sense because network management is, by definition, a network-wide activity. Logically centralized routing controllers[12,47,77] were made possible by the emergence of open-source routing software[9,40,64] that lowered the barrier to creating prototype implementations. The advances in server technology meant that a single commodity server could store all of the routing state and compute all of the routing decisions for a large ISP network.[12,79] This, in turn, enabled simple primary-backup replication strategies, where backup servers store the same state and perform the same computation as the primary server, to ensure controller reliability.

Intellectual contributions. The initial attempts to separate the control and data planes were relatively pragmatic, but they represented a significant conceptual departure from the Internet's conventional tight coupling of path computation and packet forwarding. The efforts to separate the network's control and data planes resulted in several concepts that have been carried forward in subsequent SDN designs:

• ***Logically centralized control using an open interface to the data plane.*** The ForCES working group at the IETF proposed a standard, open interface to the data plane to enable innovation in control-plane software. The SoftRouter[47] used the ForCES API to allow a separate controller to install forwarding table entries in the data plane, allowing the complete removal of control functionality from the routers. Unfortunately, ForCES was not adopted by the major router vendors, which hampered incremental deployment. Rather than waiting for new, open APIs to emerge, the RCP[12,26] used an existing standard control-plane protocol (the Border Gateway Protocol) to install forwarding table entries in legacy routers, allowing immediate deployment. OpenFlow also faced similar backward compatibility challenges and constraints: in particular, the initial OpenFlow specification relied on backward compatibility with hardware capabilities of commodity switches.

• ***Distributed state management.*** Logically centralized route controllers faced challenges involving distributed state management. A logically centralized controller must be replicated to cope with controller failure, but replication introduces the potential for inconsistent state across replicas. Researchers explored the likely failure scenarios and consistency requirements. At least in the case of routing control, the controller replicas did not need a general state management protocol, since each replica would eventually compute the same routes (after learning the same topology and routing information), and transient disruptions during routing protocol convergence were acceptable even with legacy protocols.[12] For better scalability, each controller instance could be responsible for a separate portion of the topology. These controller instances could then exchange routing information with each other to ensure consistent decisions.[79] The challenges of building distributed controllers would arise again several years later in the context of distributed SDN controllers.[46,55] These controllers face the far more general problem of supporting arbitrary controller applications, requiring more sophisticated solutions for distributed state management.

Myths and misconceptions. When these new architectures were proposed, critics viewed them with healthy skepticism, often vehemently arguing that logically centralized route control would violate *fate sharing*, since the controller could fail independently from the devices responsible for forwarding traffic. Many network operators and researchers viewed separating the control and data planes as an inherently bad idea, as initially there was no clear articulation of how these networks would continue to operate correctly if a controller failed. Skeptics also worried that logically centralized control moved away from the conceptually simple model of the routers achieving distributed consensus, where they all (eventually) have a common view of network state (e.g., through flooding). In logically centralized control, each router has only a purely local view of the outcome of the route selection process.

In fact, by the time these projects took root, even the traditional distributed routing solutions already violated these principles. Moving packet-forwarding logic into hardware meant that a router's control-plane software could fail independently from the data plane. Similarly, distributed routing protocols adopted scaling techniques, such as OSPF (Open Shortest Path First) areas and BGP (Border Gateway Protocol) route reflectors, where routers in one region of a network had limited visibility into the routing information in other regions. As discussed in the next section, the separation of the control and data planes somewhat paradoxically enabled researchers to think more clearly about distributed state management: the decoupling of the control and data planes catalyzed the emergence of a state-management layer that maintains a consistent view of network state.

In search of generality. Dominant equipment vendors had little incentive to adopt standard data-

plane APIs such as ForCES, since open APIs could attract new entrants into the marketplace. The resulting need to rely on existing routing protocols to control the data plane imposed significant limitations on the range of applications that programmable controllers could support. Conventional IP routing protocols compute routes for destination IP address blocks, rather than providing a wider range of functionality (e.g., dropping, flooding, or modifying packets) based on a wider range of header fields (e.g., MAC and IP addresses, TCP and UDP port numbers), as OpenFlow does. In the end, although the industry prototypes and standardization efforts made some progress, widespread adoption remained elusive.

To broaden the vision of control- and data-plane separation, researchers started exploring clean-slate architectures for logically centralized control. The 4D project[35] advocated four main layers: the *data plane* (for processing packets based on configurable rules); the *discovery plane* (for collecting topology and traffic measurements); the *dissemination plane* (for installing packet-processing rules); and a *decision plane* (consisting of logically centralized controllers that convert network-level objectives into packet-handling state). Several groups proceeded to design and build systems that applied this high-level approach to new application areas,[16,85] beyond route control. In particular, the Ethane project[16] (and its direct predecessor, SANE[17]) created a logically centralized, flow-level solution for access control in enterprise networks. Ethane reduces the switches to flow tables that are populated by the controller based on high-level security policies. The Ethane project, and its operational deployment in the Stanford computer science department, set the stage for the creation of OpenFlow. In particular, the simple switch design in Ethane became the basis of the original OpenFlow API.

OPENFLOW AND NETWORK OPERATING SYSTEM

In the mid-2000s, researchers and funding agencies gained interest in the idea of network experimentation at scale, encouraged by the success of experimental infrastructures (e.g., PlanetLab[6] and Emulab[83]), and the availability of separate government funding for large-scale "instrumentation" previously reserved for other disciplines to build expensive, shared infrastructure such as colliders and telescopes.[52] An outgrowth of this enthusiasm was the creation of GENI (Global Environment for Networking Innovations)[33] with an NSF-funded GENI Project Office and the EU FIRE program.[29] Critics of these infrastructure-focused efforts pointed out that this large investment in infrastructure was not matched by well-conceived ideas to use it. In the midst of this, a group of researchers at Stanford created the Clean Slate Program and focused on experimentation at a more local and tractable scale: campus networks.[51]

Before the emergence of OpenFlow, the ideas underlying SDN faced a tension between the vision of fully programmable networks and pragmatism that would enable real-world deployment. OpenFlow struck a balance between these two goals by enabling more functions than earlier route controllers and building on existing switch hardware through the increasing use of merchant-silicon chipsets in commodity switches. Although relying on existing switch hardware did somewhat limit flexibility, OpenFlow was almost immediately deployable, allowing the SDN movement to be both pragmatic and bold. The creation of the OpenFlow API[51] was followed quickly by the design of controller platforms such as NOX[37] that enabled the creation of many new control applications.

An OpenFlow switch has a table of packet-handling rules, where each rule has a pattern (which matches on bits in the packet header), a list of actions (e.g., drop, flood, forward out a particular

interface, modify a header field, or send the packet to the controller), a set of counters (to track the number of bytes and packets), and a priority (to disambiguate between rules with overlapping patterns). Upon receiving a packet, an OpenFlow switch identifies the highest-priority matching rule, performs the associated actions, and increments the counters.

Technology push and use pull. Perhaps the defining feature of OpenFlow is its adoption in industry, especially as compared with its intellectual predecessors. This success can be attributed to a perfect storm of conditions among equipment vendors, chipset designers, network operators, and networking researchers. Before OpenFlow's genesis, switch-chipset vendors such as Broadcom had already begun to open their APIs to allow programmers to control certain forwarding behaviors. The decision to open the chipset provided the necessary impetus to an industry that was already clamoring for more control over network devices. The availability of these chipsets also enabled a much wider range of companies to build switches, without incurring the substantial cost of designing and fabricating their own data-plane hardware.

The initial OpenFlow protocol standardized a data-plane model and a control-plane API by building on technology that switches already supported. Specifically, because network switches already supported fine-grained access control and flow monitoring, enabling OpenFlow's initial set of capabilities on a switch was as easy as performing a firmware upgrade—vendors did not need to upgrade the hardware to make their switches OpenFlow-capable.

OpenFlow's initial target deployment scenario was campus networks, meeting the needs of a networking research community that was actively looking for ways to conduct experimental work on "clean-slate" network architectures within a research-friendly operational setting. In the late 2000s, the OpenFlow group at Stanford led an effort to deploy OpenFlow test beds across many campuses and demonstrate the capabilities of the protocol both on a single campus network and over a wide-area backbone network spanning multiple campuses.[34]

As real SDN use cases materialized on these campuses, OpenFlow began to take hold in other realms, such as data-center networks, where there was a distinct need to manage network traffic at a large scale. In data centers, hiring engineers to write sophisticated control programs to run over large numbers of commodity switches proved to be more cost-effective than continuing to purchase closed, proprietary switches that could not support new features without substantial engagement with the equipment vendors. As vendors began to compete to sell both servers and switches for data centers, many smaller players in the network equipment marketplace embraced the opportunity to compete with the established router and switch vendors by supporting new capabilities such as OpenFlow.

Intellectual contributions. Although OpenFlow embodied many of the principles from earlier work on the separation of control and data planes, its rise offered several additional intellectual contributions:

• ***Generalizing network devices and functions.*** Previous work on route control focused primarily on matching traffic by destination IP prefix. In contrast, OpenFlow rules could define forwarding behavior on traffic flows based on any set of 13 different packet headers. As such, OpenFlow conceptually unified many different types of network devices that differ only in terms of which header fields they match and which actions they perform. A router matches on destination IP prefix and forwards out a link, whereas a switch matches on a source MAC address (to perform MAC learning) and a destination MAC address (to forward), and either floods or forwards out a single link.

Network address translators and firewalls match on a 5-tuple (source and destination IP addresses, source and destination port numbers, and transport protocol) and either rewrite address and port fields or drop unwanted traffic. OpenFlow also generalized the rule-installation techniques, allowing anything from proactive installation of coarse-grained rules (i.e., with "wild cards" for many header fields) to reactive installation of fine-grained rules, depending on the application. Still, OpenFlow does not offer data-plane support for deep packet inspection or connection reassembly; as such, OpenFlow alone cannot efficiently enable sophisticated middlebox functionality.

• ***The vision of a network operating system.*** In contrast to earlier research on active networks that proposed a node operating system, the work on OpenFlow led to the notion of a network operating system.[37] A network operating system is software that abstracts the installation of state in network switches from the logic and applications that control the behavior of the network. More generally, the emergence of a network operating system offered a conceptual decomposition of network operation into three layers: (1) a data plane with an open interface; (2) a state management layer that is responsible for maintaining a consistent view of network state; and (3) control logic that performs various operations depending on its view of network state.[46]

• ***Distributed state management techniques.*** Separating the control and data planes introduces new challenges concerning state management. Running multiple controllers is crucial for scalability, reliability, and performance, yet these replicas should work together to act as a single, logically centralized controller. Previous work on distributed route controllers[12,79] addressed these problems only in the narrow context of route computation. To support arbitrary controller applications, the work on the Onix[46] controller introduced the idea of a network information base—a representation of the network topology and other control state shared by all controller replicas. Onix also incorporated past work in distributed systems to satisfy the state consistency and durability requirements. For example, Onix has a transactional persistent database backed by a replicated state machine for slowly changing network state, as well as an in-memory distributed hash table for rapidly changing state with weaker consistency requirements. More recently, ONOS (Open Network Operating System) offers an open-source controller with similar functionality, using existing open-source software for maintaining consistency across distributed state and providing a network topology database to controller applications.[55]

Myths and misconceptions. One myth concerning SDN is that the first packet of every traffic flow must go to the controller for handling. Indeed, some early systems such as Ethane[16] worked this way, since they were designed to support fine-grained policies in small networks. In fact, SDN in general, and OpenFlow in particular, do not impose any assumptions about the granularity of rules or whether the controller handles any data traffic. Some SDN applications respond only to topology changes and coarse-grained traffic statistics, and infrequently to update rules in response to link failures or network congestion. Other applications may send the first packet of some larger traffic aggregate to the controller but not a packet from every TCP or UDP connection.

A second myth about SDN is that the controller must be physically centralized. In fact, Onix[46] and ONOS[55] demonstrate that SDN controllers can—and should—be distributed. Wide-area deployments of SDN, as in Google's private backbone,[44] have many controllers spread throughout the network.

Finally, a commonly held misconception is that SDN and OpenFlow are equivalent; in fact, OpenFlow is merely one (widely popular) instantiation of SDN principles. Different APIs could be used to control network-wide forwarding behavior; previous work that focused on routing (using

BGP as an API) could be considered one instantiation of SDN, for example, and architectures from various vendors (e.g., Cisco ONE and JunOS SDK) are other instantiations of SDN that differ from OpenFlow.

In search of control programs and use cases. Despite the initial excitement surrounding SDN, it is worth recognizing that it is merely a tool that enables innovation in network control. SDN neither dictates how that control should be designed nor solves any particular problem. Rather, researchers and network operators now have a platform at their disposal to help address longstanding problems in managing their networks and deploying new services. Ultimately, the success and adoption of SDN will depend on whether it can be used to solve pressing problems in networking that were difficult or impossible to solve with earlier protocols. SDN has already proved useful for solving problems related to network virtualization, as described in the next section.

NETWORK VIRTUALIZATION

Network virtualization, the abstraction of a network that is decoupled from the underlying physical equipment, was a prominent early use case for SDN. It allows multiple virtual networks to run over a shared infrastructure, and each virtual network can have a much simpler (more abstract) topology than the underlying physical network. For example, a VLAN (virtual local area network) provides the illusion of a single LAN spanning multiple physical subnets, and multiple VLANs can run over the same collection of switches and routers. Although network virtualization is conceptually independent of SDN, the relationship between these two technologies has become much closer in recent years.

There are three caveats to this discussion of network virtualization. First, a complete history of network virtualization would require a separate survey; this article focuses on developments that relate directly to innovations in programmable networking.

Second, although network virtualization has gained prominence as a use case for SDN, the concept predates modern-day SDN and has in fact evolved in parallel with programmable networking. The two technologies are in fact tightly coupled: programmable networks often presumed mechanisms for sharing the infrastructure (across multiple tenants in a data center, administrative groups in a campus, or experiments in an experimental facility) and supporting logical network topologies that differ from the physical network, both of which are central tenets of network virtualization.

Finally, a precise definition of network virtualization is elusive, and experts naturally disagree as to whether some of the mechanisms (e.g., slicing) represent forms of network virtualization. This article defines the scope of network virtualization to include any technology that facilitates hosting a virtual network on an underlying physical network infrastructure.

Network virtualization before SDN. For many years, network equipment has supported the creation of virtual networks in the form of VLANs and virtual private networks, but only network administrators could create these virtual networks, which were limited to running the existing network protocols. As such, incrementally deploying new technologies proved difficult. Instead, researchers and practitioners resorted to running overlay networks, where a small set of upgraded nodes use tunnels to form their own topology on top of a legacy network. In an overlay network, the upgraded nodes run their own control-plane protocol and direct data traffic (and control-plane messages) to each other by encapsulating packets, sending them through the legacy network, and de-

encapsulating them at the other end. The Mbone (for multicast),[50] 6bone (for IPv6),[43] and X-Bone[76] were prominent early examples.

These early overlay networks consisted of dedicated nodes that ran the special protocols, in the hope of encouraging adoption of proposed enhancements to the network infrastructure. The notion of overlay networks soon expanded to include any end-host computer that installs and runs a special application, spurred by the success of early peer-to-peer file-sharing applications (e.g., Napster and Gnutella). In addition to significant research on peer-to-peer protocols, the networking community reignited research on using overlay networks as a way of improving the network infrastructure, such as the work on Resilient Overlay Networks,[4] where a small collection of communicating hosts forms an overlay that reacts quickly to network failures and performance problems.

In contrast to active networks, overlay networks did not require any special support from network equipment or cooperation from ISPs, making them much easier to deploy. To lower the barrier for experimenting with overlay networks, researchers began building virtualized experimental infrastructures such as PlanetLab[60] that allowed multiple researchers to run their own overlay networks over a shared and distributed collection of hosts. Interestingly, PlanetLab itself was a form of "programmable router/switch" active networking, but using a collection of servers rather than the network nodes and offering programmers a conventional operating system (namely, Linux). These design decisions spurred adoption by the distributed-systems research community, leading to a significant increase in the role of experimentation with prototype systems in this community.

Based on the success of shared experimental platforms in fostering experimental systems research, researchers started advocating the creation of shared experimental platforms that pushed support for virtual topologies that can run custom protocols inside the underlying network,[7,61] thus enabling realistic experiments to run side by side with operational traffic. In this model, the network equipment "hosts" the virtual topology, harkening back to the early Tempest architecture, where multiple virtual ATM networks could coexist on the same set of physical switches.[78] Tempest even allowed switch-forwarding behavior to be defined using software controllers, foreshadowing the work on control- and data-plane separation.

The GENI initiative[33,59] took the idea of a virtualized and programmable network infrastructure to a much larger scale, building a national experimental infrastructure for research in networking and distributed systems. Moving beyond experimental infrastructure, some researchers argued that network virtualization could form the basis of a future Internet that allows multiple network architectures to coexist (each optimized for different applications or requirements or run by different business entities) and evolve over time to meet changing needs.[27,61,72,87]

The Relationship of Network Virtualization to SDN. Network virtualization (an abstraction of the physical network in terms of a logical network) clearly does not require SDN. Similarly, SDN (the separation of a logically centralized control plane from the underlying data plane) does not imply network virtualization. Interestingly, however, a symbiosis between network virtualization and SDN has emerged, which has begun to catalyze several new research areas. SDN and network virtualization relate in three main ways:

• ***SDN as an enabling technology for network virtualization.*** Cloud computing brought network virtualization to prominence, because cloud providers need a way of allowing multiple customers (or *tenants*) to share network infrastructure. Nicira's NVP (Network Virtualization Platform)[53] offers this abstraction without requiring any support from the underlying networking hardware. It uses

overlay networking to provide each tenant with the abstraction of a single switch connecting all of its virtual machines. In contrast to previous work on overlay networks, however, each overlay node is actually an extension of the physical network—a software switch (such as Open vSwitch[58,62]) that encapsulates traffic destined for virtual machines running on other servers. A logically centralized controller installs the rules in these virtual switches to control how packets are encapsulated, and it updates these rules when virtual machines move to new locations.

• ***Network virtualization for evaluating and testing SDNs.*** The ability to decouple an SDN control application from the underlying data plane makes it possible to test and evaluate SDN control applications in a virtual environment before they are deployed on an operational network. Mininet[41,48] uses process-based virtualization to run multiple virtual OpenFlow switches, end hosts, and SDN controllers—each as a single process on the same physical (or virtual) machine. The use of process-based virtualization allows Mininet to emulate a network with hundreds of hosts and switches on a single machine. In such an environment a researcher or network operator can develop control logic and easily test it on a full-scale emulation of the production data plane; once the control plane has been evaluated, tested, and debugged, it can be deployed on the real production network.

• ***Virtualizing ("slicing") SDN.*** In conventional networks, virtualizing a router or switch is complicated, because each virtual component needs to run its own instance of control-plane software. In contrast, virtualizing a "dumb" SDN switch is much simpler. The FlowVisor[67] system enables a campus to support a test bed for networking research on top of the same physical equipment that carries the production traffic. The main idea is to divide traffic-flow space into *slices* (a concept introduced in earlier work on PlanetLab[60]), where each slice has a share of network resources and is managed by a different SDN controller. FlowVisor runs as a hypervisor, speaking OpenFlow to each of the SDN controllers and to the underlying switches. Recent work has proposed slicing control of home networks to allow different third-party service providers (e.g., smart-grid operators) to deploy services on the network without having to install their own infrastructure.[87] More recent work proposes ways to present each slice of SDN with its own logical topology[1,22] and address space.[1]

Myths and misconceptions. People often give SDN credit for supposed benefits—such as amortizing the cost of physical resources or dynamically reconfiguring networks in multitenant environments—that actually come from network virtualization. Although SDN facilitates network virtualization and may thus make some of these functions easier to realize, it is important to recognize that the capabilities that SDN offers (i.e., the separation of data and control plane, and abstractions for distributed network state) do not directly provide these benefits.

Exploring a broader range of use cases. Although SDN has enjoyed some early practical successes and certainly offers much-needed technologies to support network virtualization, more work is needed both to improve the existing infrastructure and to explore SDN's potential to solve problems for a much broader set of use cases. Although early SDN deployments focused on university campuses,[34] data centers,[53] and private backbones,[44] recent work explores applications and extensions of SDN to a broader range of network settings, including home networks, enterprise networks, Internet exchange points, cellular core networks, cellular and Wi-Fi radio access networks, and joint management of end-host applications and the network. Each of these settings introduces many new opportunities and challenges that the community will explore in the years ahead.

CONCLUSION

The idea of a programmable network initially took shape as active networking, which espoused many of the same visions as SDN but lacked both a clear use case and an incremental deployment path. After the era of active-networking research projects, the pendulum swung from vision to pragmatism, in the form of separating the data and control planes to make the network easier to manage. This work focused primarily on better ways to route network traffic—a much narrower vision than previous work on active networking.

Ultimately, the work on OpenFlow and network operating systems struck the right balance between vision and pragmatism. This work advocated network-wide control for a wide range of applications, yet relied only on the existing capabilities of switch chipsets. Backward compatibility with existing switch hardware appealed to many equipment vendors clamoring to compete in the growing market in data-center networks. The balance of a broad, clear vision with a pragmatic strategy for widespread adoption gained traction when SDN found a compelling use case in network virtualization.

As SDN continues to develop, its history has important lessons to teach. First, SDN technologies will live or die based on "use pulls." Although SDN is often heralded as the solution to all networking problems, it is worth remembering that it is just a tool for solving network-management problems more easily. SDN merely gives developers the power to create new applications and find solutions to longstanding problems. In this respect, the work is just beginning. If the past is any indication, the development of these new technologies will require innovation on multiple timescales, from long-term bold visions (such as active networking) to near-term creative problem solving (such as the operationally focused work on separating the control and data planes).

Second, the balance between vision and pragmatism remains tenuous. The bold vision of SDN advocates a variety of control applications; yet OpenFlow's control over the data plane is confined to primitive match-action operations on packet-header fields. The initial design of OpenFlow was driven by the desire for rapid adoption, not first principles. Supporting a wide range of network services would require much more sophisticated ways of analyzing and manipulating traffic (e.g., deep-packet inspection, as well as compression, encryption, and transcoding of packets), using commodity servers (e.g., x86 machines) or programmable hardware (e.g., FPGAs, network processors, and GPUs), or both. Interestingly, the renewed interest in more sophisticated data-plane functionality, such as NFV, harkens back to the earlier work on active networking, bringing the story full circle.

Maintaining SDN's bold vision requires more thinking outside the box about the best ways to program the network without being constrained by the limitations of current technologies. Rather than simply designing SDN applications with the current OpenFlow protocols in mind, developers should think about what kind of control they want to have over the data plane, and balance that vision with a pragmatic strategy for deployment.

ACKNOWLEDGMENTS

We thank Mostafa Ammar, Ken Calvert, Martin Casado, Russ Clark, Jon Crowcroft, Ian Leslie, Larry Peterson, Nick McKeown, Vyas Sekar, Jonathan Smith, Kobus van der Merwe, and David Wetherall for detailed comments, feedback, insights, and perspectives on this article.

REFERENCES

1. Al-Shabibi, A. 2013. Programmable virtual networks: from network slicing to network virtualization; http://www. slideshare.net/nvirters/virt-july2013meetup.
2. Alexander, D. S,, Arbaugh, W. A., Keromytis, A. D., Smith, J. M. 1998. Secure active network environment architecture: realization in SwitchWare. *IEEE Network Magazine* (May): 37–45.
3. Alexander, D. S., Arbaugh, W. A., Hicks, M. W., Kakkar, P., Keromytis, A. D., Moore, J. T., Gunter, C. A., Nettles, S. M., Smith, J. M. 1998. The SwitchWare active network architecture. *IEEE Network* 12(3): 29–36.
4. Andersen, D. G., Balakrishnan, H., Kaashoek, M. F., Morris, R. 2001. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*: 131–145.
5. Anwer, M. B., Motiwala, M., bin Tariq, M., Feamster, N. 2010. SwitchBlade: a platform for rapid deployment of network protocols on programmable hardware. In *Proceedings of ACM SIGCOMM*.
6. Bavier, A. C., Bowman, M., Chun, B. N., Culler, D. E., Karlin, S., Muir, S., Peterson, L. L., Roscoe, T., Spalink, T., Wawrzoniak, M. 2004. Operating system support for planetary-scale network services. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*.
7. Bavier, A., Feamster, N., Huang, M., Peterson, L., Rexford, J. 2006. In VINI veritas: realistic and controlled network experimentation. In *Proceedings of ACM SIGCOMM*.
8. Bhattacharjee, S., Calvert, K.L.., Zegura, E. W. 1997. An architecture for active networks. In *Proceedings of High-Performance Networking*.
9. BIRD Internet routing daemon; http://bird.network.cz/.
10. Biswas, J., Lazar, A. A., Huard, J.-F., Lim, K., Mahjoub, S., Pau, L.-F., Suzuki, M., Torstensson, S., Wang, W., Weinstein, S. 1998. The IEEE P1520 standards initiative for programmable network interfaces. *IEEE Communications Magazine* 36(10): 64–70.
11. Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., Horowitz, M. 2013. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *Proceedings of ACM SIGCOMM*: 99-110.
12. Caesar, M., Feamster, N., Rexford, J., Shaikh, A., van der Merwe, J. 2005. Design and implementation of a routing control platform. In *Proceedings of the 2nd Usenix Symposium on Networked Systems Design and Implementation (NSDI)*.
13. Calvert, K. L., Ed. 1999. An architectural framework for active networks (version 1.0); http://www.cc.gatech.edu/projects/canes/papers/arch-1-0.pdf.
14. Calvert, K. 2006. Reflections on network architecture: an active networking perspective. *ACM SIGCOMM Computer Communications Review* 36(2): 27–30.
15. Calvert, K., Bhattacharjee, S., Zegura, E., Sterbenz, J. 1998. Directions in active networks. *IEEE Communications Magazine* 36(10): 72–78.
16. Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., Shenker, S. 2007. Ethane: taking control of the enterprise. In *Proceedings of ACM SIGCOMM*.
17. Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., Shenker, S. 2006. SANE: a protection architecture for enterprise networks. In *Proceedings of the 15th Usenix Security Symposium*.
18. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.

2003. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review* 33(3): 3–12.

19. Ciosi, M., et al. 2012. Network functions virtualization. Technical report, ETSI, Darmstadt, Germany; http://portal.etsi.org/NFV/NFV_White_Paper.pdf.
20. da Silva, S., Yemini, Y., Florissi, D. 2001. The NetScript active network system. *IEEE Journal on Selected Areas in Communications* 19(3): 538–551.
21. Dobrescu, M., Egi, N., Argyraki, K., Chun, B.-G., Fall, K., Iannaccone, G., Knies, A., Manesh, M., Ratnasamy, S. 2009. RouteBricks: exploiting parallelism to scale software routers. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*.
22. Drutskoy, D., Keller, E., Rexford, J. 2013. Scalable network virtualization in software-defined networks. *IEEE Internet Computing* 17(2): 20-27.
23. Erickson, D. 2013. The Beacon OpenFlow controller. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software-defined Networking (HotSDN)*.
24. Erickson, D., et al. 2008. A demonstration of virtual machine mobility in an OpenFlow network. Demonstration at ACM SIGCOMM. http://yuba.stanford.edu/~nickm/papers/p513-ericksonA.pdf
25. Farrel, A., Vasseur, J.,-P. Ash, J. 2006. A Path Computation Element (PCE)-based architecture. Internet Engineering Task Force RFC 4655. https://tools.ietf.org/html/rfc4655
26. Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., van der Merwe, K. 2004. The case for separating routing from routers. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*.
27. Feamster, N., Gao, L., Rexford, J. 2007. How to lease the Internet in your spare time. *ACM SIGCOMM Computer Communications Review* 37(1): 61–64.
28. Floodlight OpenFlow Controller; http://floodlight.openflowhub.org/.
29. FlowVisor; http://www.openflowswitch.org/wk/index.php/FlowVisor.
30. Fraser, A. G. 1980. Datakit—a modular network for synchronous and asynchronous traffic. In *Proceedings of the International Conference on Communications*.
31. Future Internet Design. National Science Foundation; http://www.nets-find.net/.
32. Future Internet Research and Experimentation Initiative. European Union; http://www.ict-fire.eu/.
33. GENI: Global Environment for Network Innovations; http://www.geni.net/.
34. GENI. 2011. Campus OpenFlow topology; http://groups.geni.net/geni/wiki/OpenFlow/CampusTopology.
35. Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., Zhang, H. 2005. A clean-slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review* 35(5): 41–54.
36. Greene, K. 2009. TR10: software-defined networking. *MIT Technology Review* (March/April); http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/.
37. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S. 2008. NOX: Towards an operating system for networks. *ACM SIGCOMM Computer Communication Review* 38(3): 105–110.
38. Han, S., Jang, K., Park, K., Moon, S. 2010. PacketShader: a GPU-accelerated software router. In *Proceedings of ACM SIGCOMM*.
39. Handigol, N., Flajslik, M., Seetharaman, Johari, R. 2010. S. McKeown, N., Aster*x: load-balancing as a network primitive. In *Proceedings of Architectural Concerns in Large Datacenters (ACLD)*.

40. Handley, M., Kohler, E., Ghosh, A., Hodson, O., Radoslavov, P. 2005. Designing extensible IP router software. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*.
41. Heller, B., Handigol, N., Jeyakumar, V., Lantz, B., McKeown, N. 2012. Reproducible network experiments using container based emulation. In *Proceedings of ACM CoNEXT*.
42. Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., McKeown, N. 2010. ElasticTree: saving energy in data-center networks. Presented at 7th Usenix Symposium on Networked Systems Design and Implementation (NSDI); https://www.usenix.org/legacy/event/nsdi10/tech/full_papers/heller.pdf.
43. Hinden, R., Postel, J. 1996. IPv6 testing address allocation. Internet Engineering Task Force RFC 1897, made obsolete by RFC 2471 on 6bone Phaseout. https://tools.ietf.org/html/rfc1897
44. Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A. 2013. B4: experience with a globally deployed software-defined WAN. In *Proceedings of ACM SIGCOMM*.
45. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M. F. 2000. The Click modular router. *ACM Transactions on Computer Systems* 18(3): 263–297.
46. Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., Shenker, S. 2010. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th Usenix Symposium on Operating Systems Design and Implementation*: 351-364; https://www.usenix.org/legacy/event/osdi10/tech/full_papers/osdi10_proceedings.pdf.
47. Lakshman, T. V., Nandagopal, T., Ramjee, R., Sabnani, K., Woo, T. 2004. The SoftRouter architecture. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets)*; http://conferences.sigcomm.org/hotnets/2004/HotNets-III%20Proceedings/lakshman.pdf.
48. Lantz, B., Heller, B., McKeown, N. 2010. A network in a laptop: rapid prototyping for software-defined networks (at scale!). In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks (HotNets)*.
49. Lockwood, J. W., McKeown, N., Watson, G., Gibb, G., Hartke, P., Naous, J. Raghuraman, R., Luo, J. 2007. NetFPGA: an open platform for gigabit-rate network switching and routing. In *IEEE International Conference on Microelectronic Systems Education*: 160–161.
50. Macedonia, M. R., Brutzman, D. P. 1994. Mbone provides audio and video across the Internet. *Computer* 27(4): 30–36.
51. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J. 2008. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38(2): 69-74.
52. National Science Foundation. NSF Guidelines for Planning and Managing the Major Research Equipment and Facilities Construction (MREFC) Account. 2005; http://www.nsf.gov/bfa/docs/mrefcguidelines1206.pdf.
53. Nayak, A., Reimers, A., Feamster, N., Clark, R. 2009. Resonance: dynamic access control in enterprise networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*.
54. Nicira. It's time to virtualize the network. 2012; http://www.netfos.com.tw/PDF/Nicira/It%20is%20Time%20To%20Virtualize%20the%20Network%20White%20Paper.pdf.
55. ON.Lab. 2013. ONOS: Open Network Operating System; http://www.slideshare.net/umeshkrishnaswamy/open-network-operating-system.

56. Open Daylight; http://www.opendaylight.org/.
57. Open Networking Foundation; https://www.opennetworking.org/.
58. Open vSwitch; openvswitch.org.
59. Peterson, L., Anderson, T., Blumenthal, D., Casey, D., Clark, D., Estrin, D., Evans, J., Raychaudhuri, D., Reiter, M., Rexford, J., Shenker, S., Wroclawski, J. 2006. GENI design principles. *IEEE Computer* 39(9): 102–105.
60. Peterson, L., Anderson, T., Culler, D., Roscoe, T. 2002. A blueprint for introducing disruptive technology into the Internet. In *Proceedings of the 1st ACM Workshop on Hot Topics in Networks (HotNets).*
61. Peterson, L., S. Shenker, S., Turner, J. 2004. Overcoming the Internet impasse through virtualization. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets).*
62. Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T., Shenker, S. 2009. Extending networking into the virtualization layer. In *Proceedings of the 8th ACM Workshop on Hot Topics in Networks (HotNets).*
63. POX; http://www.noxrepo.org/pox/about-pox/.
64. Quagga software routing suite; http://www.quagga.net/.
65. Salim, J., Khosravi, H., Kleen, A., Kuznetsov, A. 2003. Linux Netlink as an IP services protocol. Internet Engineering Task Force, RFC 3549. https://tools.ietf.org/html/rfc3549
66. Schwartz, B., Jackson, A. W., Strayer, W. T., Zhou, W., Rockwell, R. D., Partridge, C. 1999. Smart packets for active networks. In *Proceedings of the 2nd IEEE Conference on Open Architectures and Network Programming:* 90–97.
67. Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., Parulkar, G. 2010. Can the production network be the testbed? In *Proceedings of the 9th Usenix Symposium on Operating Systems Design and Implementation (OSDI).*
68. Smith, J., et al. 1996. SwitchWare: accelerating network evolution. Technical Report MS-CIS-96-38, University of Pennsylvania.
69. Smith, J. M., Calvert, K. L., Murphy, S. L., Orman, H. K., Peterson, L. L. 1999. Activating networks: a progress report. *Computer* 32(4): 32–41.
70. Smith, J. M., Nettles, S. M. 2004. Active networking: one view of the past, present, and future. *IEEE Transactions on Systems, Man, and Cybernetics: Part C: Applications and Reviews* 34(1).
71. Spalink, T., Karlin, S., Peterson, L., Gottlieb, Y. 2001. Building a robust software-based router using network processors. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP):* 216-229.
72. Taylor, D., Turner, J. 2005. Diversifying the Internet. In *Proceedings of the IEEE Global Telecommunications Conference.*
73. Taylor, D. E., Turner, J. S., Lockwood, J. W., Horta, E. L. 2002. Dynamic hardware plug-ins: exploiting reconfigurable hardware for high-performance programmable routers. *Computer Networks* 38(3): 295–310.
74. Tennenhouse, D., Smith, J., Sincoskie, W. D., Wetherall, D., Minden, G. 1997. A survey of active network research. *IEEE Communications Magazine* 35(1): 80–86.
75. Tennenhouse, D. L., Wetherall, D. J. 1996. Towards an active network architecture. *ACM SIGCOMM Computer Communication Review* 26(2):5–18.
76. Touch, J. 2001. Dynamic Internet overlay deployment and management using the X-Bone. *Computer Networks* 36(2): 117–135.

77. van der Merwe, J., Cepleanu, A., D'Souza, K., Freeman, B., Greenberg, A., et al. 2006. Dynamic connectivity management with an intelligent route service control point. In *ACM SIGCOMM Workshop on Internet Network Management.*
78. van der Merwe, J., Rooney, S., Leslie, I., Crosby, S. 1998. The Tempest: a practical framework for network programmability. *IEEE Network* 12(3): 20–28.
79. Verkaik, P., Pei, D., Scholl, T., Shaikh, A., Snoeren, A., van der Merwe, J. 2007. Wresting control from BGP: scalable fine-grained route control. In *Proceedings of the Usenix Annual Technical Conference*.
80. Voellmy, A., Hudak, P. 2011. Nettle: functional reactive programming of OpenFlow networks. In *Proceedings of the Workshop on Practical Aspects of Declarative Languages*: 235–249.
81. Wang, R., Butnariu, D., Rexford, J. 2011. OpenFlow-based server load balancing gone wild. In *Proceedings of the Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE).*
82. Wetherall, D., Guttag, J., Tennenhouse, D. 1998. ANTS: a toolkit for building and dynamically deploying network protocols. In *Proceedings of IEEE OpenArch*.
83. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A. 2002. An integrated experimental environment for distributed systems and networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI).*
84. Wolf, T., Turner, J. S. 2001. Design issues for high-performance active routers. *IEEE Journal on Selected Areas of Communication* 19(3): 404–409.
85. Yan, H., Maltz, D. A., Ng, T. S. E., Gogineni, H., Zhang, H., Cai, Z. 2007. Tesseract: a 4D network control plane. In *Proceedings of the 4th Usenix Symposium on Network Systems Design and Implementation (NSDI).*
86. Yang, L., Dantu, R., Anderson, T., Gopal, R. 2004. Forwarding and Control Element Separation (ForCES) Framework. Internet Engineering Task Force, RFC 3746. https://www.rfc-editor.org/rfc/rfc3746.txt
87. Yiakoumis, Y., Yap, K.-K., Katti, S., Parulkar, G., McKeown, N. 2011. Slicing home networks. In *ACM SIGCOMM Workshop on Home Networking (Homenets).*
88. Zander, J., Forchheimer, R. 1983. Softnet: an approach to higher-level packet radio. In *Proceedings of the Amateur Radio Computer Networking Conference*.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

JENNIFER REXFORD is the Gordon Y.S. Wu professor of engineering in the computer science department at Princeton University. Before joining Princeton in 2005, she worked for eight years at AT&T Labs—Research. She received a B.S.E. in electrical engineering from Princeton in 1991, and a Ph.D. in electrical engineering and computer science from the University of Michigan in 1996. She served as the chair of ACM SIGCOMM from 2003 to 2007, and was the 2004 winner of ACM's Grace Murray Hopper Award for outstanding young computer professional.

NICK FEAMSTER is an associate professor in the College of Computing at Georgia Tech. He received a Ph.D. in computer science from MIT in 2005, and S.B. and M.Eng. degrees in electrical engineering

and computer science from MIT in 2000 and 2001, respectively. His research focuses on many aspects of computer networking and networked systems. In December 2008, he received the Presidential Early Career Award for Scientists and Engineers (PECASE) for his contributions to cybersecurity, notably spam filtering. His honors include the Technology Review 35 "Top Young Innovators Under 35" award, the ACM SIGCOMM Rising Star Award, and the IRTF (Internet Research Task Force) Applied Networking Research Prize.

ELLEN W. ZEGURA is Professor of Computer Science in the College of Computing at Georgia Tech, where she has held a faculty position since 1993. Her interests are in computer networking, with a focus on the Internet and mobile wireless networks, and on humanitarian computing. She is a Fellow of the ACM and a Fellow of the IEEE. All of her degrees are from Washington University in St. Louis.