Software defined networking to support the software defined environment

C. Dixon
D. Olshefski
V. Jain
C. DeCusatis
W. Felter
J. Carter
M. Banikazemi
V. Mann
J. M. Tracey
R. Recio

Software defined networking (SDN) represents a new approach in which the decision-making process of the network is moved from distributed network devices to a logically centralized controller, implemented as software running on commodity servers. This enables more automation and optimization of the network and, when combined with software defined compute and software defined storage, forms one of the three pillars of IBM's software defined environment (SDE). This paper provides an overview of SDN, focusing on several technologies gaining attention and the benefits they provide for cloud-computing providers and end-users. These technologies include (i) logically centralized SDN controllers to manage virtual and physical networks, (ii) new abstractions for virtual networks and network virtualization, and (iii) new routing algorithms that eliminate limitations of traditional Ethernet routing and allow newer network topologies. Additionally, we present IBM's vision for SDN, describing how these technologies work together to virtualize the underlying physical network infrastructure and automate resource provisioning. The vision includes automated provisioning of multi-tier applications, application performance monitoring, and the enabling of dynamic adaptation of network resources to application workloads. Finally, we explore the implications of SDN on network topologies, quality of service, and middleboxes (e.g., network appliances).

Introduction

Software defined networking (SDN) represents a fundamental advancement, revolutionizing the network industry [1]. SDN differs from traditional networking in three critical ways. First, SDN separates the data plane, which forwards traffic at full speed, from the control plane, which makes decisions about how to forward traffic at longer time scales. Second, SDN provides a well-defined interface between the now-separated control and data planes, including a set of abstractions for network devices that hide the many of their details. Third, SDN migrates control plane logic to a logically centralized controller that exploits a global view of network resources and knowledge of application requirements to implement and optimize global policies. **Figure 1** shows a high-level SDN architecture where the SDN controller provides the centralized control plane functionality for the switches and allows SDN applications to provide network functionality. As a result of these three seemingly simple changes to traditional network architectures, SDN should radically increase the pace of network innovation and improve the performance, scalability, cost, flexibility, security, and ease of management. Along with software defined compute and software defined storage, IBM's software defined environment (SDE) allows for the automation and optimization of entire computing environments providing similar benefits.

At its core, SDN allows a high-level abstract specification of network connectivity and services to be automatically and dynamically mapped to a set of underlying network resources. **Figure 2** depicts an abstract representation of the network for a three-tier web workload. The thick lines represent connectivity between the tiers, whereas thin

Digital Object Identifier: 10.1147/JRD.2014.2300365

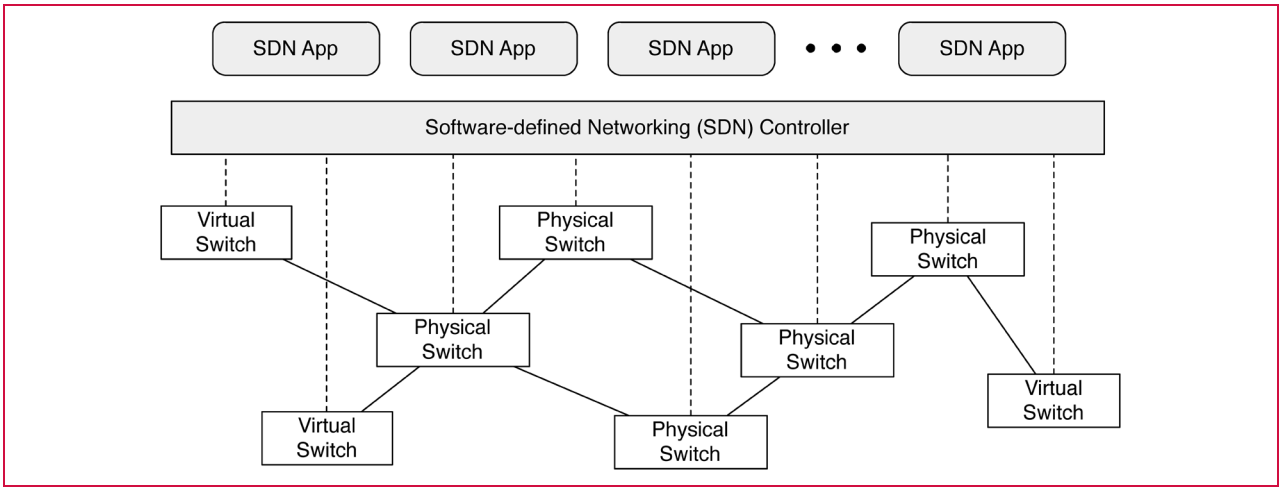

Figure 1

A high-level view of a standard SDN network architecture where a (logically) centralized SDN controller provides the control plane for all of the switches. The solid lines represent the physical connectivity between switches (either physical or virtual) and the dotted lines represent communication between the controller and the switches to install forwarding rules. The controller provides new network functionality by hosting SDN applications.

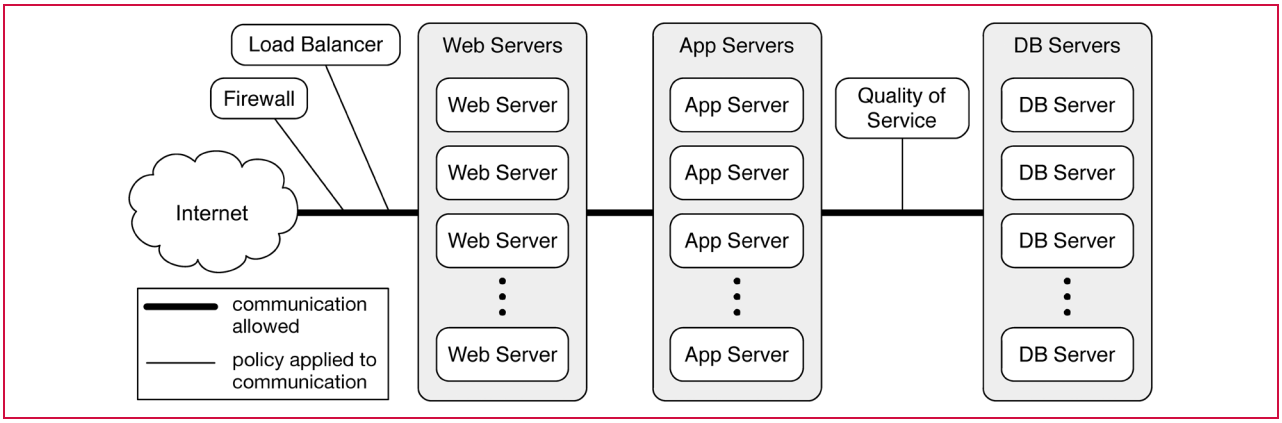

Figure 2

The abstract network specification for a three-tier web workload including a firewall service and a load balancer service on the logical link between the Internet and web servers. Further, connections between application servers and database servers are labeled with a higher quality of service.

lines indicate policy and/or network services applied to connectivity. **Figure 3** shows the underlying network resources for one potential realization of the abstract network depicted in Figure 2. The thick arrows correspond to the abstract connectivity in Figure 2, whereas thin lines indicate the physical links in the network. The underlying implementation is far more complicated than the abstract representation. Traditional networks are specified in terms of the underlying implementation. This is analogous to programming in assembly language. Software defined networks are specified in abstract terms that more closely resemble programming in a high-level language.

Virtualization and abstraction

SDN defines open, standard abstractions for networks that hide the details of the underlying infrastructure, similar to how an operating system abstracts the complexity of underlying hardware by exporting common application programming interfaces (APIs) to services such as file systems, virtual memory, sockets, and threads. These

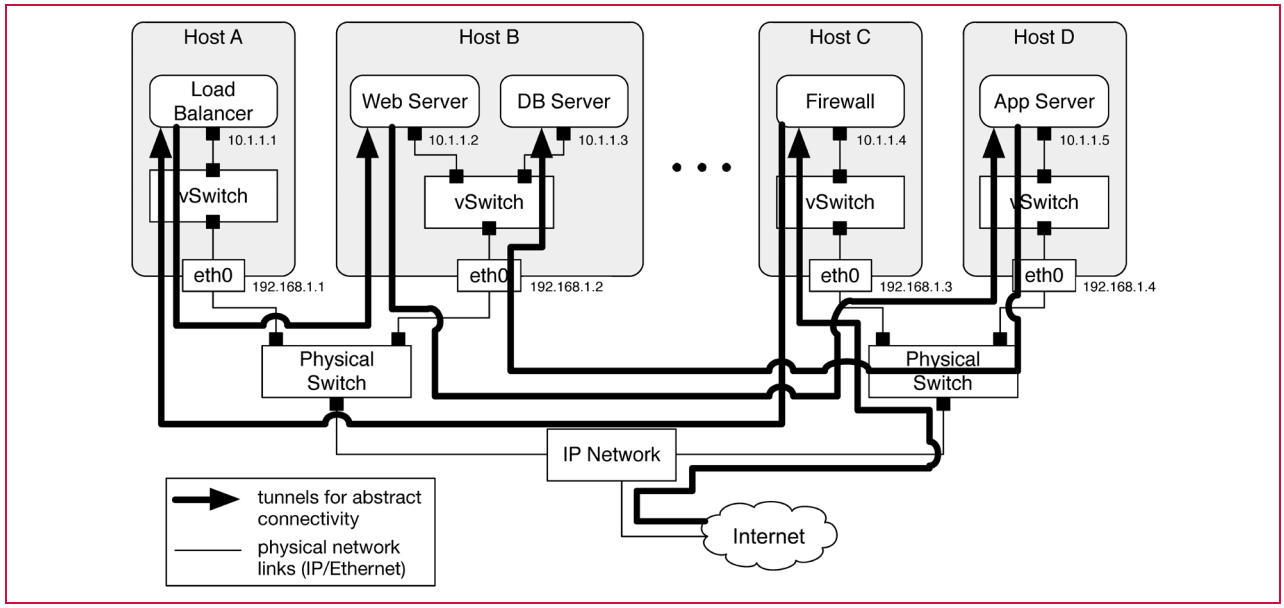

Figure 3

The underlying network implementing the abstract network shown in Figure 2. The thick arrows represent the tunnels that provide the abstract connectivity from the Internet to the various services via the appropriate middleboxes. The hypervisor virtual switches (labeled vSwitch) are responsible for routing traffic in to and out of the tunnels. The quality of service requirements between the application servers and database servers are provided by configuration in the switches and are not shown.

abstractions provide new tools to enable the richer networking functionalities demanded by recent industry trends including dynamic virtual server workloads, multi-tenant cloud computing, and warehouse-scale data centers. Existing standards and abstractions have proven inadequate for delivering this functionality "at scale"; for example, 12-bit virtual local area network (VLAN) identifiers [2] allow for up to 4,096 isolated tenants. To address this, networks have become increasingly complex, including proprietary routing, traffic engineering mechanisms, and labor-intensive configuration of network appliances used to secure and optimize multi-tier systems. SDN offers the potential to reverse this trend by addressing these problems in the controller software running on commodity servers that programs network hardware using open protocols.

The dominant use of SDN that enables solutions to these problems is *network virtualization*. Network virtualization involves abstracting the physical network in two ways: (i) isolating multiple tenants and giving them a "view" such that they are the only ones using the network and (ii) presenting an abstract topology that may differ from the physical topology, e.g., an abstract topology with all hosts attached to a single, large switch. A related concept is Network Functions Virtualization (NFV) [3], which replaces specialized appliances such as firewalls, load balancers, and intrusion detection systems with virtual machines (VMs) running on conventional servers [4–7] connected to the network. In the server world, virtualization has enabled new applications and revenue streams that would not have been technically possible or economically feasible otherwise. It is anticipated the same will be true for networking.

Splitting the data plane and the control plane

In conventional networks, each device implements both data and control plane functionality. For example, when a packet is received at a switch, the data plane matches fields in the packet with respect to forwarding rules and performs specified actions such as changing the destination Internet Protocol (IP) address and forwarding the packet on a specific port. To instantiate these rules, various mechanisms are used. For basic forwarding, each device participates in distributed control plane logic, communicating with peers using protocols such as spanning tree [8] or Transparent Interconnection of Lots of Links (TRILL) [9] for Ethernet switches—and Border Gateway Protocol (BGP) [10] or Open Shortest Path First (OSPF) [11] for IP routers.

SDN combines these control channels into one mechanism that can control both basic forwarding and more sophisticated services. Each device continues to forward packets at full speed on the basis of currently installed forwarding rules, but the distributed control plane is replaced with a logically centralized controller that programs the forwarding rules of each device in the network. The controller uses its global network view to create basic forwarding rules that are not

limited to spanning trees and dovetail with higher-level functionalities such as Network Address Translation (NAT) and VLANs. The ability to control all aspects of the network results in flexibility and innovation.

Centralizing network control

Once the data and control planes are split, it is no longer necessary to have a distributed control plane. As a consequence most realizations of SDN migrate a substantial portion of network control functionality to a logically centralized SDN controller. The controller connects to every switch in the network, typically through a separate control network, which allows it to monitor and control each device. A tightly coupled cluster of SDN controllers can be used for scale-out and fault-tolerance [12–14]. In such clusters, consistently distributing shared state can be problematic and many recent efforts have explicitly distributed some tasks either to a subset of the cluster or to switches to alleviate these issues [15, 16]. Though less common, the distributed management plane can also be replaced with a logically centralized management point, possibly the same controller, to enable network-wide monitoring, management, and policy enforcement [17].

While there are well-recognized trade-offs between distributed and centralized control, the advantages of centralization appear to greatly outweigh the disadvantages in the context of SDN. Most of the problems described earlier can be solved using SDN technology. For example, an SDN controller has global visibility into the current state of the network, e.g., link and buffer utilization, device failures, and where hosts are located, so it can implement end-to-end quality of service (QoS) and respond rapidly to failures [18, 19]. However, SDN need not centralize control *entirely*. Internet scale networks and networks of large organizations will continue to consist of numerous independent administrative domains.

The rest of this paper is organized as follows. The next section describes several example SDN scenarios: network virtualization and abstraction, middlebox connectivity, fabrics, monitoring-control loops, and QoS. That is followed by a discussion of IBM's SDN vision and offerings including the OpenDaylight** Project [20]. After that, examples of early adopters of SDN technology are presented. Two final sections briefly discuss migration to SDN technology and provide concluding remarks.

Example SDN scenarios

SDN provides a platform on which a wide variety of scenarios can be realized. Ultimately, the functionality delivered via this platform will be limited only by developers' ingenuity and imagination. Here, we describe six scenarios that have already been realized: network virtualization, network abstraction, middlebox insertion, fabrics, monitoring-control loops, and QoS. These examples are chosen to both give an idea of the breadth of SDN's uses and illustrate some of the most prevalent current use cases.

Network virtualization

Perhaps the most common SDN scenario is network virtualization. In fact, people sometimes mistakenly equate SDN with virtualization. The Merriam-Webster dictionary defines virtual as "being such in essence or effect though not formally recognized or admitted." For example, a virtual Ethernet switch, such as Open vSwitch [21], behaves just like a physical Ethernet switch. It receives incoming Ethernet frames on its ports and forwards them, transmitting each on the appropriate port(s), based on fields in the frame header such as the destination address. Unlike a physical Ethernet switch, however, a virtual switch is realized in software, typically in a hypervisor running on a general-purpose computer. This enables virtual switches, and virtual networks, to be created quickly and easily, which is one of the key benefits of virtualization.

Virtual Ethernet switches forward Ethernet frames between virtual network interface cards (vNICs) associated with VMs running on a hypervisor and one or more of the hypervisor's physical Ethernet NICs. This allows a single physical Ethernet NIC to be shared by multiple VMs. By itself, interface sharing only allows vNICs to be connected to physical Ethernet switches. A typical physical Ethernet network consists of multiple interconnected switches. To construct a virtual network, the connections between switches must be virtualized as well. This raises some interesting challenges for which SDN is particularly well-suited.

Virtual networks can be implemented using a variety of means. One is by using VLAN tags. Such tags segment traffic flowing throughout a single Ethernet into multiple conceptual or "virtual" local area networks. Switches direct traffic tagged for a particular VLAN only to ports associated with the corresponding tag thus enforcing the appearance of multiple and separate networks. At first glance, VLANs might appear to be an ideal mechanism for network virtualization. Their effectiveness for this purpose is limited by several factors. First, VLANs do not, by themselves, span multiple Ethernet networks, or even multiple Ethernet switches. Instead, they require careful coordination of the configuration at all switches that might need to handle traffic for that VLAN. This is a severe limitation for a virtual network, which can reasonably be expected to span multiple physical networks potentially over geographically dispersed regions. Second, the total number of VLANs is limited to 4,096 because of the number of bits in the VLAN tag. This limit is smaller than the number of virtual networks that might be needed in even a small to medium sized data center. VLANs, though useful, do not provide an ideal solution for network virtualization.

A more effective approach to network virtualization is to use an *overlay network*. With this approach, the virtual switches *encapsulate* or "wrap" virtual Ethernet frames in one or more higher layer network protocols, typically User Datagram Protocol (UDP) and/or IP. Virtual switches are connected to one another via IP "tunnels" for which there are two leading and competing standards: Virtual eXtensible Local Area Network (VXLAN) [22] and Network Virtualization using Generic Routing Encapsulation (NVGRE) [23].

The tunnels form a network that is *overlaid* on top of an IP network. This approach has several benefits. One is the lack of any need to configure or otherwise control physical switches. The only requirement overlays have on the underlying network is IP connectivity, which has long been ubiquitous. Another benefit is their significantly greater scalability relative to VLANs. One important consideration with overlays is the approach used to select switches between which a tunnel is established. Tunnels could be established between every pair of switches. This generates n^2 tunnels where n is the number of virtual switches. In another approach, a single virtual switch is selected with which all other switches establish a tunnel. Both approaches can run into scalability issues when there are very large numbers of virtual switches. Current solutions use traditional routing, e.g., BGP or OSPF, to interconnect different overlay networks when they reach these scaling limits.

Overlays also have some drawbacks. Perhaps the largest is that the mappings from destination addresses to corresponding tunnels have to be disseminated and kept up-to-date even as VMs join, leave and move. This adds either increased latency to the first packet while the virtual switch pulls this information from a centralized service or extra communication and complexity overhead to keep correct entries pre-cached. Further, encapsulation adds additional headers reducing the amount of data in each physical Ethernet frame available for useful data (slightly) decreasing throughput. Encapsulation and de-encapsulation also entail processing and memory overhead. Finally, while building on top of IP eases deployment, it also means accepting IP's best effort delivery. Alternately, the overlay controller can leverage QoS features in the underlying switches at the cost of the added complexity involved in configuring and using QoS queues in each switch.

A third approach to network virtualization leverages OpenFlow** [24]. OpenFlow is a standard protocol by which an SDN controller can configure forwarding rules in Ethernet switches. The protocol allows packets to be identified by a comprehensive set of header fields such as source and destination Media Access Control (MAC) addresses, IP addresses and Transmission Control Protocol (TCP) or UDP port numbers. OpenFlow can be used to implement a virtual network by programming virtual and physical switches to forward packets between network interfaces attached to the same virtual network while prohibiting flows between different networks. OpenFlow does not require encapsulation. Furthermore, once a switch is programmed for a particular flow, it forwards frames using the same mechanism and identical performance as for a conventional nonvirtualized network.

A key consideration when leveraging OpenFlow is the approach used to program switch forwarding rules. Rules may be programmed *reactively*, where packets that do not match an already-installed rule are sent to the controller, which then installs an appropriate rule. They may also be programmed *proactively*, for example, when a new host is discovered or a virtual network is defined. This exposes a tradeoff between the added latency of contacting the controller on the first packet of each flow and needing to keep rules installed for all flows versus only currently active flows. The reactive approach is more commonly used today as the number of fully flexible rules that can be simultaneously installed in physical switches is often quite limited, e.g., 1,000 rules. Increasing rule sizes and hybrid approaches promise to improve this state of affairs.

OpenFlow provides fine-grained control over a switch's forwarding behavior including the ability to enforce QoS. This can be leveraged to limit or eliminate performance interference between distinct virtual networks. OpenFlow too has its limitations. Its primary drawback is that it requires OpenFlow capable switches and the administrative authority to exploit that capability. OpenFlow is a new and still emerging technology so its deployment is currently limited. Currently deployed OpenFlow controllers typically manage a limited set of switches such as those within a single data center. Thus, while highly effective for use within a data center, OpenFlow, by itself does not currently represent a viable solution for network virtualization across geographically dispersed sites without using overlay tunnels between "islands" of OpenFlow switches.

The preceding discussion of virtualization options may seem to reflect increasing complexity rather than the simplification promised by SDN. However, the details of the implementation and ultimately the selection of an implementation are handled by the SDN controller. A virtual network user need only specify the required connectivity and desired network services. The controller takes these specifications and configures implementation artifacts such as tunnels or OpenFlow rules as needed to realize virtual networks on the available underlying network resources.

Network abstraction

Traditionally, one measure of virtualization quality is the extent to which virtual entities appear to be as real as possible. Fidelity between real and virtual entities allows the entire ecosystem of software and expertise developed in the real world to be leveraged in the virtual one. It eliminates the need for solutions already perfected on real resources

to be re-implemented. One benefit of virtualization however is the ability of virtual entities to offer *enhanced* functionality. SDN therefore provides not only network virtualization, but also abstraction. The Merriam-Webster dictionary defines the term *abstract* as "disassociated from any specific instance." An abstract network, therefore, is one that retains the fundamental capabilities of a general class of networks, for example packet forwarding, without exhibiting the specific behaviors of any one instance, such as a network of Ethernet switches.

An abstract network adheres to a given *model*. The model defines the entities supported by the network and their behaviors including interactions among entities and with the external world. Many models are possible. Virtual Ethernet, described earlier, though not particularly abstract, represents one network model. Virtual Ethernet implementations typically provide some level of abstraction. One popular model is the "big switch" in which all virtual network interfaces appear to be plugged into a single switch regardless of the presence of multiple underlying physical and virtual Ethernet switches. A single link in a virtual Ethernet may actually correspond to multiple links or paths in the physical network. Higher levels of abstraction are possible. Ethernet follows a "default on" approach. In the absence of specific action to the contrary, all interfaces connected to the same network can communicate with one another.

An alternate model from IBM Research called *Meridian* [25] takes a "default off" approach. With Meridian, specifying that server A can communicate with both server B and server C does not imply B and C can communicate with each other. This facilitates handling real world scenarios that are not as well suited to a default on model. One example is where a set of web servers must communicate with a set of application servers but there is no requirement for communication among the web or application servers themselves.

As with high level programming languages, numerous models are possible and model development is expected to continue for the foreseeable future. A good model is generally easy to use, abstract, dynamic, comprehensive and composable. A model should be intuitive to use and naturally fit typical scenarios. It should not mandate specification of details that may not always be necessary. A good model is adept at handling changes in both the abstract and physical networks. Strong models provide comprehensive network services, for example address assignment and name resolution, enabling but not requiring essential services be provided externally. Finally, a good model facilitates composition of higher-level services from fundamental model entities.

An SDN controller may support multiple models concurrently. These models may exhibit *equivalence* in which any scenario described in one can be identically represented in another. In this case, it may be possible to interact with a single abstract network instance using multiple models. Incompatibilities between models may also exist. In such cases, the controller may augment or adapt one or more models to enable better interoperation. Alternately, such incompatibilities may simply prohibit an abstract network specified using one model from being manipulated via another.

Middlebox insertion

A middlebox is a hardware or virtual appliance that provides functionality along a network path. Example middleboxes include firewalls, intrusion detection and/or prevention, proxies, caches, and wide area network (WAN) accelerators. Middlebox deployment is a significant pain point for data centers. Physical appliances are expensive to purchase and require a specialized skill set to configure thus increasing both capital and operational expense. They often have to be cabled in-line at specific locations in the network limiting flexibility. Firmware upgrades can be a risky undertaking. To date, the industry has failed to converge on a standard protocol for configuration of firewalls, load balancers and other middleboxes. Data centers are left with a variety of heterogeneous appliances, each with their own method of configuration whose semantics differ from device to device.

An emerging trend is replacement of physical middleboxes with virtual appliances, most recently termed NFV. Virtual appliances are highly flexible in that they can be placed on any host server. Being VMs, they can be copied and started throughout the data center making them ideal for auto-scaling in response to traffic load. One approach taken in this direction is to create a distributed platform for middlebox appliances (e.g., vAMOUR [26] or Embrane** [27]). The middlebox platform controller is separate from the SDN controller and deploys the required middlebox functionality across the VMs it manages. The middlebox controller tunnels traffic between middlebox VMs as specified by a user-specified chain. A drawback to this approach is that, typically, a given middlebox controller only interoperates with middleboxes from that vendor either limiting the set of available middleboxes or requiring multiple such controllers.

IBM's vision is for the SDN controller to manage middlebox deployment as an integral part of virtual network deployment, and to consider middleboxes as first class entities within the network model. In actuality, we would like to manage network services, which might be middleboxes or other network functionality, such as higher QoS. The user can specify which types of cloud provided network services need to service which flows. The user may accept the default rule set that is provided by a firewall or specify some custom changes in a manner that declares the desired semantics without having to delve into a low-level rule specification language.

To support middleboxes from different vendors, the SDN controller provides a plug-in mechanism that enables vendors to manage their company's middleboxes using their vendor-specific protocol. The SDN controller maps the virtual network model, including its network services, to an internal representation and calls the appropriate vendor plug-in to configure the appropriate middleboxes. The SDN controller manages the routing of all packet flows in the network—between hosts and network services. This integrated, global view and control allows for fast scale up/down, insertion and removal of network services and servers.

Current virtualized middleboxes mimic their physical counterparts and thus inherit certain drawbacks. This includes the need for network interfaces into each network that they service. Overlay network based service chaining can aid in removing these restrictions thereby simplifying the deployment of these devices.

Network fabrics

A fabric is a set of network devices, such as switches, that provide connectivity as a holistically managed unit. Fabrics are a natural fit for SDN because a controller normally manages a set of switches in much the same way. Fabrics typically provide shortest path forwarding and some degree of multipathing. They can thus offer significantly more bandwidth than traditional networks—especially at the Ethernet layer where the spanning tree protocol typically restricts these features. Fabrics also usually allow for straightforward host mobility, a feature missing from traditional IP networks since moving between subnets requires changing a host's IP address.

While there are recent standards that build fabrics in a distributed manner [9, 28], with SDN fabrics are constructed by having the logically centralized controller determine the network topology, e.g., via Link Level Discovery Protocol (LLDP) [29], and then directly compute and install routes. This centralized approach can offer faster convergence to network changes, e.g., failures, and globally optimized use of network resources. A variety of such SDN fabrics have been proposed and some built [30–33] offering different trade-offs. IBM's SDN controller implements a fabric called Scalable Per-Address RouTing Architecture (SPARTA) based on PAST (Per-Address Spanning Tree) [34]. A key benefit of SDN fabrics is the ability to improve, or replace, the fabric simply by upgrading the SDN controller.

As fabrics need not implement standard routing protocols internally, they open the possibility of using more exotic topologies than the typical folded-Clos-style fat trees used today. There are a variety of existing topologies optimized for different communication patterns, e.g., HyperX [35], the cabling-optimized Dragonfly [36], and the randomly wired Jellyfish [37]. These topologies provide multiple shortest paths between endpoints, so fabrics' support for multiple paths can exploit these unconventional topologies to increase performance and/or lower cost. These new topologies typically employ a single type of switch instead of the different core, aggregation, and edge switches commonly found in current networks. Thus, an entire fabric can be built out of a large number of relatively inexpensive top-of-rack switches.

When a fabric has multiple paths available, it must choose which path to use for each flow. Typically, switches hash packet header fields and use the hash function to choose the path; this spreads different flows onto different paths while preventing packets of a single flow from being reordered. Hashing works well when there are a large number of flows of roughly equal bandwidth, but high-bandwidth flows and random hash collisions cause load imbalances within fabrics. Advanced fabrics can monitor for such imbalances and reroute flows to spread load more evenly, reducing congestion and increasing overall network throughput. SDN makes such automatic traffic engineering easier to implement because the central controller can aggregate measurements of the whole network and perform global optimizations [18, 19].

Monitoring-control loops

While the most significant work on SDN has come on the control side, the ability to program switches is worthless without the context of how they are connected and what traffic they must forward.

To provide this context, SDN solutions typically provide three monitoring mechanisms:

1. *Topology discovery*—Mechanisms to discover links between network devices and thus deduce the topology of the network. This includes both physical links, and multi-hop links, e.g., tunnels.
2. *New flow detection*—A way to get notifications of new flows where a flow can be defined as a new host joining, a new host-pair communicating, or a new TCP/UDP connection depending on the circumstances.
3. *Counters*—Counters track the number of bytes and packets handled at different granularities, e.g., per switch, per host, or per port.

These mechanisms provide an unprecedented level of automated monitoring. Rather than painstakingly maintaining an authoritative diagram of network topology, the network itself can produce the diagram more accurately. New flow detection and counters provide feedback about network utilization in near real-time. This allows for both human operators and automated tools to understand what their networks are doing and why at time scales and fidelities previously impossible.

Polling counters on all network devices from a centralized SDN controller may prove to be costly. Prior work has shown that to be effective, the polling should finish within a second or less [38, 39], which is difficult. Several researchers have explored using flow monitoring techniques such as NetFlow [40] and sFlow [41] in conjunction with topology discovery [42]. Nearly all current virtual and physical switches support NetFlow or sFlow. NetFlow enabled switches send Netflow records to a *collector* at periodic intervals. These contain details about all flows observed during the last monitoring interval. sFlow enabled switches sample one out of every N packets (where N is a configured sampling rate) and sends the packet header to a collector. These approaches offer potentially lower-latency and more scalable approaches to monitoring the current network state. Significant prior work has focused on how to use these measurements to effectively infer network conditions [43] and perform high layer functions such as network aware VM management [44].

Combining accurate knowledge of the network topology, flows, and link utilization with the ability to change forwarding behavior illustrates the strength of SDN. This combination allows an adaptive control loop to monitor and react to network events in real time. For example, this can be used to redirect large flows away from congested links.

Over time, with proper annotation of particular workloads, the controller can even learn the network behavior of different workloads and adapt the forwarding rules in anticipation rather than merely as a reaction. In a virtualized environment, VM placement can be based on network conditions and VMs can be migrated if beneficial. At even longer time-scales, knowledge of fine-grained workload usage combined with big data analytics can inform decisions of how to reconfigure and upgrade the network. For example, simply adding an extra link between certain racks might have the same effect for certain workloads as upgrading the entire network.

Quality of service

SDN enables several types of network QoS. A coarse form of "best-effort" QoS can be achieved by quickly detecting congestion and rerouting traffic around congested links. This is an active area of research and several researchers have looked at how to reduce time required to detect congestion and reroute flows [45].

Providing stronger QoS guarantees typically requires two switch features: (i) the ability to create and configure queues with dedicated bandwidth at each switch port and (ii) the ability to place flows into these queues [46]. The latter is easily available through the OpenFlow enqueue action, whereas the former has been largely achieved out-of-band. Configuration of queues is an essential part of the OF-Config specification [15]. An alternate approach proposed recently is to combine rate control at end hosts with OpenFlow-based control of flows in the physical network to give end-to-end guarantees [47].

In an environment, where SDN is supported only at the network edge, QoS options are limited. A controller can employ explicit rate control (supported in Open vSwitch), but without knowledge of the internal topology it is difficult or impossible to avoid oversubscribing links. Alternately, the controller can explicitly set the IP type of service [48] and/or Ethernet priority code point [49] packet header fields for downstream routers and switches to act on.

IBM's SDN approach, architecture, and offerings

IBM SDN for Virtual Environments (SDN-VE) is an SDN controller that can control an overlay network, a fabric, or both together. The SDN-VE controller is based on the Linux Foundation's OpenDaylight Project [20]. Alongside the controller, IBM provides virtual switches designed for specific hypervisor platforms including virtual switch for VMware (the IBM System Networking's Distributed Virtual Switch 5000V [50]) and a virtual bridge for Kernel-based VM (KVM) as part of SDN-VE.

This section describes SDN-VE including its OpenDaylight base, the architectural framework, how that framework integrates with cloud infrastructure such as OpenStack, and operates over various underlying protocols such as OpenFlow.

OpenDaylight

IBM was a founding member of the OpenDaylight Project, a Linux Foundation project chartered to create a unified, open source SDN controller and other technologies. IBM's most significant contribution to date is the Open Distributed Overlay Virtual Ethernet (Open DOVE) network virtualization component based on earlier research at IBM [51]. The project's first release includes support for a wide range of network protocols including OpenFlow 1.0, OpenFlow 1.3, Path Computation Element Communication Protocol (PCEP) [52], Border Gateway Protocol (BGP) [10], the Locator/ID Separation Protocol (LISP) [53], and services including Distributed Denial of Service (DDoS) mitigation and network virtualization. At its core, the OpenDaylight controller consists of Java**-based, OSGi bundles [54] that allow for individual components of the controller to be easily swapped in and out combined with a Service Abstraction Layer (SAL) that insulates applications or services running on top of the controller from the protocol details of "drivers" that logically sit at the bottom of the controller.

Architectural framework

The SDN-VE architecture consists of: (i) the controller core, northbound APIs and southbound driver plug-in interfaces; (ii) integrated network services and applications,

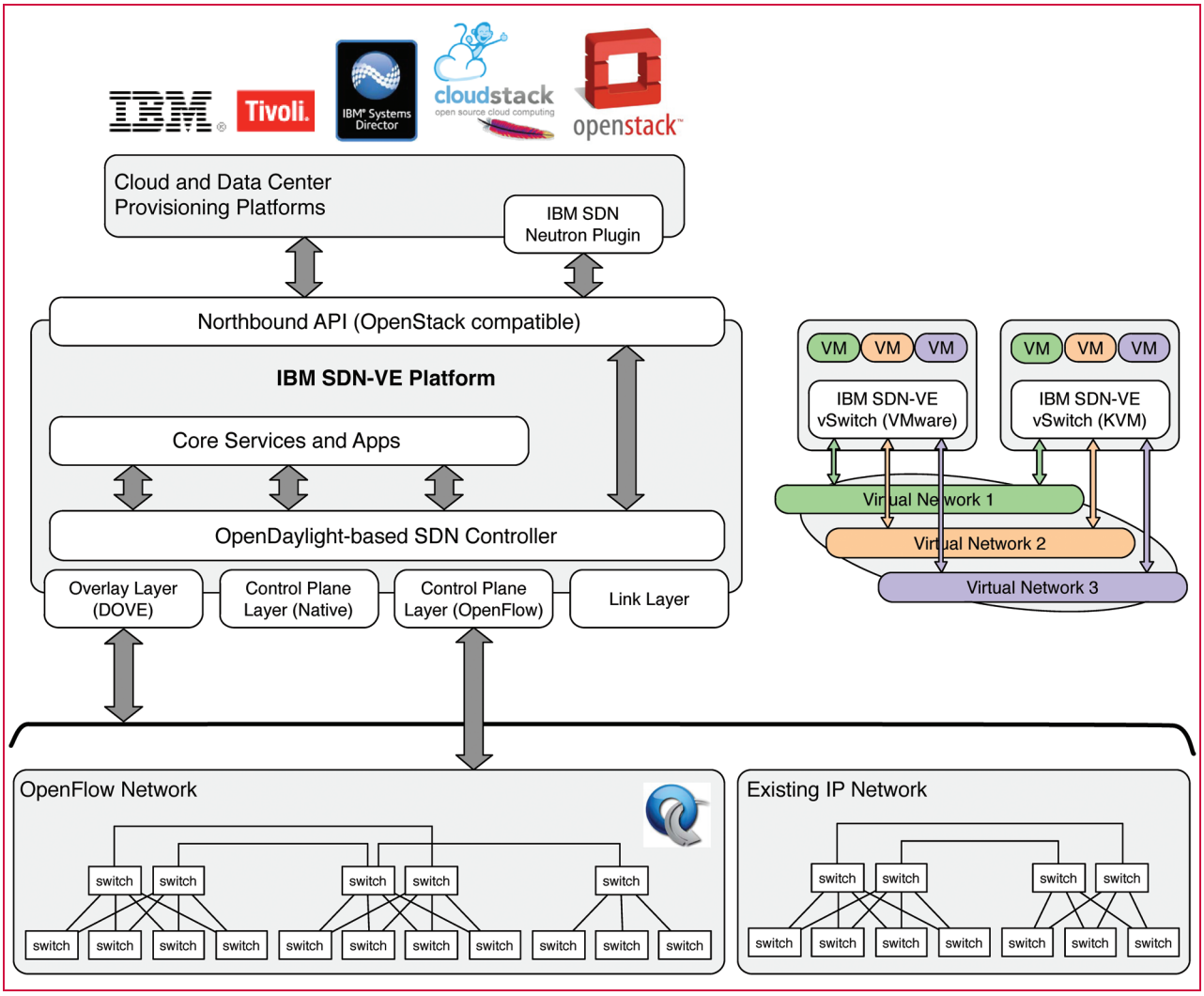

Figure 4

The IBM SDN-VE architecture. The core platform consists of three layers: (i) the abstract, northbound network API, which is compatible with OpenStack Neutron, (ii) an orchestration layer (represented by "Core Services and Apps" and the "OpenDaylight-based SDN controller," which maps abstract API calls to actions on devices and vice versa, and (iii) the southbound drivers for devices including OpenFlow and DOVE. Provisioning platforms make use of the northbound APIs to configure the network. The DOVE driver provides overlay virtual networks (shown on the middle right) across both OpenFlow and IP networks (shown at the bottom), while the OpenFlow driver allows for control of the underlying fabric if it supports OpenFlow.

which can be hosted within or outside the SDN-VE platform; and (iii) the underlying physical network. The SDN-VE platform has a layered architecture. Network services and applications sit at the top layer and interact with the controller via an abstract network API. The middle layer is an orchestration layer that interprets the abstract network API calls, converts them into network specific tasks and implements the tasks through one or more drivers at the bottom layer. The bottom layer is composed of a set of protocol drivers or interfaces enabling the controller to communicate with different devices as well as deployed environments in the network. **Figure 4** shows the architecture of the SDN-VE controller.

SDN-VE controller core—The core provides a set of common functionality essential for building network services and applications. It includes an object modeling infrastructure to represent network elements such as switches, routers and gateways along with their configuration and interconnections as well as a data store

for maintaining configurations, topology, state, policies and other critical data needed for controller operation. Further, it provides synchronization between controller instances deployed in a cluster for fault-tolerance and to scale-out. In addition, the core provides methods for unified topology discovery and maintenance.

Northbound APIs—The SDN-VE northbound API is a superset of Neutron [55], the OpenStack networking API. This has been deliberately designed to make the integration of SDN with SDE, which is based on OpenStack, easier as discussed in SDN integration section below. Currently, the SDN-VE northbound API provides support layer-2 virtualization using networks, subnets, and ports as primitives. Further, it can perform layer-3 routing between subnets it manages. It also supports several Neutron extensions such as port binding and provider network [56] extensions.

Southbound drivers—The SDN-VE platform offers extensibility through southbound drivers that control network devices. In particular it uses (i) OpenFlow 1.0 and 1.3 drivers to manage switches that support those protocols and (ii) drivers for virtual switches in hypervisors to implement overlay virtualization using Distributed Overlay Virtual Ethernet (DOVE) including support for KVM [57] and VMware environments. Over time, we expect that SDN-VE will add additional drivers both from OpenDaylight and IBM efforts.

Orchestration layer—The orchestration layer is responsible for mapping requests received from northbound layer to the appropriate set of devices and consequently drivers. A similar mapping from devices to northbound API concepts translates events generated in the network to applications that have registered for such notifications.

Network applications integrated with SDN-VE—The SDN-VE platform is designed to be extensible through the addition of network applications and even possibly open to third-party developers, but a variety of core applications come pre-integrated with SDN-VE:

- *Logical networks*—This service enables creation of disparate logical networks over the shared physical infrastructure, along with provisioning for the necessary physical and virtual resources. It supports multi-tenancy with per tenant virtual network configuration, policies, traffic isolation, statistics and topology views.
- *Connectivity service*—Based on the grouping of hosts into logical networks, the connectivity determines whether hosts should be allowed to communicate and if so, through what middleboxes and/or network services.
- *Scalable Per-Address RouTing Architecture (SPARTA)*—SPARTA is a scalable layer-2 forwarding solution designed to make efficient use of network fabrics. SPARTA, which is based on PAST [34], computes per-MAC-address, destination-rooted spanning trees and programs switches to forward traffic to each host in the network along its associated tree. SPARTA supports an arbitrary (mesh) topology and balances flows across multiple paths, thus helps do away with the limitations of traditional Ethernet networks with a single spanning tree. Further, SPARTA leverages access to larger layer-2 specific forwarding tables to support more than 100,000 hosts on some switches.
- *Flow replication and redirection*—This allows a higher layer application to logically tap into flows between a given source and destination by having the traffic replicated to another destination. This service can be used for pushing selected flows to diagnostic tools. It can also test new services on live traffic before deployment.

Service plane integration—As SDN is deployed, a key aspect is how the SDN controller integrates network services and appliances. SDN-VE supports three different kinds of network services. First, the controller can route traffic through an existing physical appliance, but if the appliance is not SDN-aware, the traffic may need to be proxied by an SDN gateway that removes any SDN-specific encapsulation and headers. Second, virtual appliances behave similarly, but can be dynamically instantiated as VMs on commodity servers and proxied by the hypervisor virtual switch. Finally, some network services can be implemented purely via OpenFlow rules installed by a software module running in the controller.

To orchestrate this integration, the SDN-VE northbound API extends the Neutron API to allow for the specification of network service and middlebox insertion. In particular, when using overlay routing, SDN-VE provides the ability to define and configure special virtual networks that contain (either physical or virtual) network services such as firewalls and load balancers. Further, these services can be chained and applied to fine-grained subsets of traffic. For example, only TCP traffic on port 80 between two virtual networks. The application of these chains and the services in them is determined solely by properties of the traffic and the grouping of the source and destination into virtual networks, rather than being based on hosts' physical locations. IBM is in the process of proposing these service chaining features and APIs as additions to OpenStack Neutron.

IBM SDN integration with SDE architectures

OpenStack [58], an open source platform for cloud computing is emerging as the platform of choice for building open clouds and forms the core of IBM's SDE. The OpenStack networking component, Neutron, provides ways to create basic layer-2 and layer-3 connectivity and is quickly expanding to support higher-level network services. In particular, Neutron currently defines standard ways to specify

that firewalls, load balancers and Virtual Private Network (VPN) services should be applied to certain traffic. Both OpenDaylight and IBM's SDN-VE provide plug-ins to interface with Neutron. Further, the extensibility of Neutron allows the differentiating features of SDN-VE to be presented to OpenStack users through Neutron extensions.

While overlay network virtualization merely uses the existing physical network for IP transport, SDN-VE is also able to control the underlying fabric and coordinate actions between the underlay and overlay. For example, when SDN-VE provides a SPARTA fabric, the overlay component can ask for link-disjoint paths to avoid correlated failures. When it comes to interoperability between the fabric and/or overlay and existing physical networks, SDN-VE ensures the behavior at the edge is compatible with the layer-2 and layer-3 expectations of existing physical networks.

Early adoption

This section describes several early SDN deployments to give an idea of what real-world use cases exist and thus how others might start to deploy SDN.

Financial data distribution in real time

A provider of financial information had specific service-level agreement (SLA) requirements to deliver financial information to different subscribers while ensuring fairness so each subscriber received information at the same time. They used IBM's OpenFlow solution including IBM switches and IBM's Programmable Network Controller to move forwarding decisions off servers/firewalls and into an OpenFlow network. Defining flow rules that rewrote packets, added them to multicast groups and forwarded them to specified ports, all in a few microseconds ensured that they were able to meet the SLA requirement.

Application-driven network bandwidth allocation

A large European university deployed an SDN-based solution using IBM switches that enables network administrators to easily configure and manage virtual networks that control traffic on a per-flow basis based on application requirements. This helps to ensure more predictable performance for large transfers of data in the university's complex environment.

On-demand network monitoring via virtual taps

A large managed service provider deployed an SDN-based solution using IBM switches to provide the ability to "tap" any location in their data center network and send packets from the tap to a centralized location for analysis. This gave them the flexibility to deploy virtual monitoring points throughout the network to diagnose performance and other network issues without having to dedicate physical resources to the monitoring when it is not in use.

Careful management of expensive and critical network resources

One of the first high-profile displays of the current generation of SDN solutions was Google's announcement that it was using OpenFlow to operate its "G-Scale" private inter-data-center WAN. The SDN features of this network allow for traffic over their WAN to be scheduled and to take non-default paths enabling them to more efficiently use the expensive resource. In particular, they are able to persistently drive their WAN links to near-maximum utilization without ill effects giving a 2–3 times improvement over standard practices [59–61].

Pervasive security enforcement

One of the major changes between traditional networking and SDN is the notion that SDN holistically manages the network in a logically centralized manner. An obvious area that this can help with is security. Security appliances such as firewalls and intrusion detection/prevention systems have typically been point solutions and ensuring that they sit at choke points in the network has been error-prone and performance limiting. Instead SDN allows for a controller to dictate logically pervasive security and route traffic through the appropriate appliances or services. HP has demonstrated this approach with clients in the enterprise space in their presentation at the 2013 Open Network Summit [62]. IBM offers similar features in SDN-VE.

Migration

Given the magnitude of the shift between traditional networking and SDN, it is difficult not to wonder how to introduce SDN into an existing network, allowing for interoperation with legacy networks, and eventually migrating more of the network to SDN. While this paper elides a fuller discussion of these issues due to space limitations, we highlight some of the issues.

SDN deployments can involve an overlay, control of physical devices, or both. In general, deploying an overlay alone requires less effort as it does not usually require new hardware and instead uses existing layer-3 connectivity. Overlays do generally require new software often in the form of a specialized virtual switch for hypervisors and a controller, but this is still less invasive than deploying a full SDN solution which controls physical network devices as well.

Interoperation with existing networks requires some planning and while there are a variety of strategies, a common strategy is to use existing internetworking approaches to stitch SDN-enabled portions of the network to legacy networks. For example, having a layer-3 gateway that handles traffic to and from the SDN-enabled network as though it were a normal IP prefix or subnet reduces, but does not eliminate, the integration issues. Alternately, the SDN-enabled network can speak typical interoperability

protocols like OSPF, BGP, or Intermediate System to Intermediate System (IS-IS). Recent research efforts have also looked at how to maximally benefit from a limited deployment of SDN by using traditional network approaches to ensure that all paths traverse at least one SDN-enabled device [63].

Lastly, deploying SDN can change the set of risks a network operator or architect must take in to consideration. The SDN controller and the channels between it and network devices are new elements that need to be considered from both a security and stability standpoint. In terms of stability, a variety of recent work [64–66] has applied model checking and software verification to SDN controllers and applications to find bugs and instabilities. As for security, if the SDN controller is attached to an already-secured management network, it should offer little additional attack surface, but in some cases, especially when controlling hypervisor virtual switches, the controller must connect to the data network which makes it more critical to secure both the controller and switches. Fortunately, most SDN protocols, including OpenFlow, allow for authenticated and encrypted communication.

Conclusion

SDN promises to drastically increase the flexibility and efficiency of computer networks. By separating the forwarding plane from the control plane and moving the control plane to logically centralized software running on commodity hardware, we can rapidly innovate and provide new network features while globally optimizing network behavior. This can already be seen in current solutions, which offer the ability to easily deploy multiple virtual networks on the same physical hardware while also providing simpler, higher-level abstractions of the network to ease configuration and management. Going forward, SDN will enable more efficient use of network resources by enabling tight monitoring-control loops to perform adaptive traffic engineering, intelligent workload placement and QoS guarantees.

IBM's SDN efforts can be seen in the SDN-VE product, which allows for abstract virtual networks to be rapidly instantiated. Further, SDN-VE interoperates with OpenStack and provides the networking features of IBM's broader SDE initiative via an OpenStack Neutron plugin, which offers virtual networking APIs. It is capable of providing overlay virtual networks, managing the underlying fabric, or both in concert. It also includes a suite of network applications including middlebox interposition, flow redirection/duplication, and network-aware VM placement. Combined, these enable SDN-VE to easily manage networks for multi-tenant clouds. Going forward, SDN-VE will serve as a platform for IBM's future SDN efforts.

SDN remains a fertile ground for research and development. Remaining challenges include (i) the design and implementation of abstract network models that both allow for expressive policy and are easy to understand, (ii) developing frameworks to allow SDN controllers to be extended with third-party modules while avoiding conflicts, and (iii) more intelligent orchestration of network resources to provide self-managing and self-tuning networks.

Acknowledgments

We thank Amitabha Biswas and Uday Shankar Nagaraj for their contributions to this paper as well as the development teams that have made IBM's SDN products a reality including the SDN-VE team and the open source community surrounding OpenDaylight. We also appreciate the constructive feedback from our anonymous reviewers.

References

1. Software Defined Networking Creates a New Approach to Delivering Business Agility. [Online]. Available: http://www.gartner.com/newsroom/id/2386215
2. IEEE 802.1Q Virtual LANs (VLANs). [Online]. Available: http://www.ieee802.org/1/pages/802.1Q.html
3. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv
4. V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. NSDI*, 2012, p. 24.
5. Middlebox communication architecture and framework, IETF RFC 3303. [Online]. Available: http://www.ietf.org/rfc/rfc3303.txt
6. V. Sekar, S. Ratnasamy, M. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: Enabling innovation in middlebox deployment," in *Proc. HotNets*, 2011, p. 21.
7. J. Sherry, S. Hasan, and C. Scott, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. SIGCOMM*, Helsinki, Finland, 2012.
8. Spanning Tree Protocol (STP). [Online]. Available: http://www.cisco.com/en/US/docs/switches/lan/catalyst6500/ios/12.2SX/configuration/guide/spantree.html
9. Transparent Interconnection of Lots of Links (TRILL). [Online]. Available: http://tools.ietf.org/html/rfc5556
10. Border Gateway Protocol (BGP). [Online]. Available: http://www.ietf.org/rfc/rfc1771.txt
11. Open Shortest Path First (OSPF). [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt
12. D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized? State distribution trade-offs in software defined networks," in *Proc. Hotnets*, 2012, pp. 1–6.
13. A. Panda, C. Scotty, A. Ghodsiy, T. Koponen, and S. Shenker, "CAP for networks," in *Proc. HotSDN*, 2013, pp. 91–96.
14. T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, 2010, pp. 1–6.
15. S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. HotSDN*, 2012, pp. 19–24.
16. S. Schmid and J. Suomela, "Exploiting locality in distributed SDN control," in *Proc. HotSDN*, 2013, pp. 121–126.
17. OpenFlow Configuration and Management Protocol (OF-Config). [Online]. Available: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow-config

18. M. A. Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, 2010, p. 19.
19. T. Benson, A. Anand, A. Akkela, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. CoNEXT*, 2011, p. 8.
20. OpenDaylight | A Linux Foundation Collaborative Project. [Online]. Available: http://www.opendaylight.org/
21. Open vSwitch—An Open Virtual Switch. [Online]. Available: http://www.openvswitch.org
22. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, IETF draft draft-mahalingam-dutt-dcops-vxlan-04.txt
23. NVGRE: Network Virtualization using Generic Routing Encapsulation, IETF draft draft-sridharan-virtualization-nvgre-00.txt
24. OpenFlow Switch Specification V1.1.0, Feb. 2011. [Online]. Available: http://www.openflow.org
25. M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: An SDN platform for cloud network services," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 120–127, Feb. 2013.
26. vARMOUR. [Online]. Available: http://www.varmour.com/
27. Embrane. [Online]. Available: http://www.embrane.com/
28. IEEE 802.1aq Shortest Path Bridging (SPB). [Online]. Available: http://www.ieee802.org/1/pages/802.1aq.html
29. Link Layer Discovery Protocol (LLDP). [Online]. Available: http://www.ieee802.org/1/pages/802.1ab.html
30. A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. SIGCOMM*, 2009, pp. 51–62.
31. J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies," in *Proc. USENIX NSDI*, 2010, p. 18.
32. R. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. SIGCOMM*, 2009, pp. 39–50.
33. C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: A scalable ethernet architecture for large enterprises," in *Proc. SIGCOMM*, 2008, pp. 3–14.
34. B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable ethernet for data centers," in *Proc. CoNEXT*, 2012, pp. 49–60.
35. J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "HyperX: Topology, routing, and packaging of efficient large-scale networks," in *Proc. SC*, 2009, p. 41.
36. J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Proc. ISCA*, Jun. 2008, pp. 77–88.
37. A. Singla, C. Hong, L. Popa, and P. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. NSDI*, 2012, p. 17.
38. C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley, "Data center networking with multipath TCP," in *Proc. HotNets*, 2010, p. 10.
39. A. R. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. SIGCOMM*, 2011, pp. 254–265.
40. Netflow. [Online]. Available: www.cisco.com/go/netflow
41. sflow. [Online]. Available: http://www.sflow.org
42. V. Mann, A. Vishnoi, and S. Bidkar, "Living on the edge: Monitoring network flows at the edge in cloud data centers," in *Proc. COMSNETS*, 2013, pp. 1–9.
43. D. Arifler, G. de Veciana, and B. L. Evans, "A factor analytic approach to inferring congestion sharing based on flow level measurements," *IEEE Trans. Netw.*, vol. 15, no. 1, pp. 67–79, Feb. 2007.
44. V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware steady state VM management for data centers," in *Proc. IFIP Netw.*, 2012, pp. 190–204.
45. J. Su, T. Kwon, C. Dixon, W. Felter, and J. Carter, "OpenSample: A low-latency, sampling-based measurement platform for SDN." IBM Research Technical Report, IBM Corporation, 2014.
46. V. Mann, A. Vishnoi, A. Iyer, and P. Bhattacharya, "VMPatrol: Dynamic and automated QoS for virtual machine migrations," in *CNSM*, 2012, pp. 174–178.
47. M. Mishra, P. Dutta, P. Kumar, and V. Mann, "On managing network utility of tenants in oversubscribed clouds," in *IEEE MASCOTS*, 2013, p. 221.
48. Type of Service in the Internet Protocol Suite. [Online]. Available: http://tools.ietf.org/html/rfc1349
49. IEEE 802.1p Ethernet Priority Code Point. [Online]. Available: http://www.juniper.net/techpubs/en_US//junos/topics/concept/cos-qfx-series-lossless-ieee8021p-priority-config-understanding.html
50. IBM CorporationIBM Distributed Virtual Switch 5000V. [Online]. Available: http://www-03.ibm.com/systems/networking/switches/virtual/dvs5000v/
51. K. Barabash, R. Cohen, D. Hadas, V. Jain, R. Recio, and B. Rochwerger, "A case for overlays in DCN virtualization," in *Proc. 3rd Workshop DC CAVES*, pp. 30–37. [Online]. Available: http://www.itc23.com/fileadmin/ITC23_files/papers/DC-CaVES-m1569472213.pdf
52. Path Computation Element Communication Protocol (PCEP). [Online]. Available: http://tools.ietf.org/html/rfc5440
53. Locator/ID Separation Protocol (LISP). [Online]. Available: http://www.lisp4.net/
54. OSGi. [Online]. Available: http://www.osgi.org/
55. Neutron—Openstack. [Online]. Available: https://wiki.openstack.org/wiki/Neutron
56. Quantum provider network extensions. [Online]. Available: https://review.openstack.org/#/c/25843/
57. KVM. [Online]. Available: www.linux-kvm.org/page/Main page
58. OpenStack. [Online]. Available: http://www.openstack.org/
59. S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *Proc. SIGCOMM Conf.*, Hong Kong, 2013, pp. 3–14.
60. A. Vahdat, "SDN@Google: Why and how," presented at the Open Networking Summit, Santa Clara, CA, USA, 2013. [Online]. Available: http://www.opennetsummit.org/archives-april2013/
61. U. Hoelzle, "OpenFlow@Google," presented at the Open Networking Summit, Santa Clara, CA, USA, 2012. [Online]. Available: http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf
62. B. Mayer, "Virtual application networks innovations advance software-defined network leadership," presented at the Open Networking Summit, Santa Clara, CA, USA, 2013. [Online]. Available: http://www.opennetsummit.org/pdf/2013/presentations/bethany_mayer.pdf
63. D. Levin, M. Canini, S. Schmid, and A. Feldmann, "Toward transitional SDN deployment in enterprise networks," presented at the Open Networking Summit, Santa Clara, CA, USA, 2013. [Online]. Available: http://www.opennetsummit.org/pdf/2013/research_track/poster_papers/final/ons2013-final22.pdf
64. A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. 10th USENIX Symp. NSDI*, Apr. 2013, pp. 15–28.
65. P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. 9th USENIX Symp. NSDI*, Apr. 2012, p. 9.
66. M. Canini, D. Venzano, P. Perešíni, D. Kosti, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proc. 9th USENIX Symp. NSDI*, Apr. 2012, p. 10.

Received August 23, 2013; accepted for publication September 18, 2013

Colin Dixon *IBM Research Division, Austin Research Lab, Austin, TX 78758 USA (ckd@us.ibm.com).* Dr. Dixon is a Researcher in the Data Center Networking Group at the IBM Austin Research Lab. His recent work has focused on software-defined networking with a focus on data centers, but more generally, his research interests are broadly in systems, spanning networks, distributed systems, operating systems and security with an emphasis on building real, secure, reliable, and efficient computer systems. He earned his M.S. and Ph.D. degrees in computer science from the University of Washington in Seattle in 2007 and 2011, respectively. Previously he received a B.S. degree in mathematics and a B.S. degree in computer science from the University of Maryland in Collage Park in 2005. He has coauthored more than a dozen technical papers and articles as well as several patents and patent applications.

David Olshefski *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (olshef@us.ibm.com).* Dr. Olshefski is a Research Staff Member in the Systems Department at the IBM T. J. Watson Research Center. He received a B.S. degree in computer science from SUNY Albany, M.S. degree in computer science from RPI at Hartford, and a Ph.D. degree in computer science from Columbia University. He has coauthored more than a dozen technical papers and articles as well as several patents.

Vinit Jain *IBM Systems and Technology Group, Austin, TX 78758 USA (vjain@us.ibm.com).* Mr. Jain is the Chief SDN Solutions Engineer for IBM Systems Networking and currently leads the integration of IBM SDN products into solutions. During the past 15 years with IBM, he has led the delivery of several innovations in networking and virtualization products. He is a Master Inventor with more than 50 issued patents. Mr. Jain has a B.Tech. degree in computer science from Indian Institute of Technology, New Delhi, India, and an M.S. degree in computer science University of Maryland, College Park.

Casimer DeCusatis *IBM Systems and Technology Group, Poughkeepsie, NY 12603 USA (decusat@us.ibm.com).* Dr. DeCusatis is an IBM Distinguished Engineer and CTO for Strategic Alliances, System Networking Division. He received the M.S. and Ph.D. degrees from Rensselaer Polytechnic Institute (Troy, NY) in 1988 and 1990, respectively, and the B.S. degree *magna cum laude* in the Engineering Science Honors Program from the Pennsylvania State University (University Park, PA) in 1986. He is an IBM Master Inventor with more than 120 patents and has published more than 100 technical papers. He has received the IEEE Kiyo Tomiyasu Award, the Electronic Device News Innovator of the Year Award, the Mensa Research Foundation Creative Achievement award, the Sigma Xi Walston Chubb award, the Penn State Outstanding Scholar Alumni award, and the IEEE/Eta Kappa Nu Outstanding Young Electrical Engineer award (including a citation from the President of the United States and an American flag flown in his honor over the U.S. Capitol). He is editor of the *Handbook of Fiber Optic Data Communication* (now in its fourth edition), a member of the Order of the Engineer, and a Fellow of the IEEE, Optical Society of America, and SPIE, the International Society of Optical Engineering.

Wes Felter *IBM Research Division, Austin Research Lab, Austin, TX 78758 USA (wmf@us.ibm.com).* Mr. Felter is a researcher in the data-center networking group at the IBM Austin Research Lab. His current research focus involves building scalable low-cost network fabrics, and in the past, he worked extensively on server and storage power management. He received a B.S. degree in computer science from the University of Texas at Austin in 2001.

John Carter *IBM Research Division, Austin Research Lab, Austin, TX 78758 USA (retrac@us.ibm.com).* Dr. Carter leads the Future Systems department of IBM Research - Austin, which includes teams investigating next generation data center network architectures, enterprise mobile and cloud runtimes and services, and energy-efficient server, storage, and memory system design. His personal research has spanned many areas of systems research, including distributed systems (Munin and Khazana), multiprocessor computer architecture (Avalanche and S-COMA), advanced memory system design (Ultraviolet and Impulse), and networking (SPARTA). Dr. Carter received his Ph.D. degree from Rice University in 1993. He spent 15 years on the faculty of the School of Computing at the University of Utah and several years as Chief Scientist of MangoSoft before joining IBM Research. He has published over 60 conference and journal articles and written several dozen patents, and is a Senior Member of the IEEE and ACM.

Mohammad Banikazemi *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (mb@us.ibm.com).* Dr. Banikazemi is a Research Staff Member in the Systems Department at the IBM T. J. Watson Research Center. His research interests include cloud computing and software-defined networking. He has an M.S. degree in Electrical Engineering and a Ph.D. degree in computer science both from the Ohio State University, where he was an IBM Graduate Fellow. He is an author or coauthor of several technical papers and patents. He is a Senior Member of IEEE and ACM.

Vijay Mann *IBM Research - India, New Delhi, India (vijamann@in.ibm.com).* Mr. Mann is a senior software engineer and manager at IBM Research - India in New Delhi. He currently manages the data center networking team at IBM Research - India. This is his tenth year at IBM Research - India. In the past, he has worked on systems for portfolio analytics at Morgan Stanley, New York. He has more than 12 years of experience in enterprise systems development and research. He has authored more than 20 publications in well-known conferences and journals and has several filed and issued patents to his credit. He holds an M.S. degree from Rutgers University, New Jersey, and a B.E. degree from Malaviya National Institute of Technology (MNIT), Jaipur, India.

John M. Tracey *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (traceyj@us.ibm.com).* Dr. Tracey is a Senior Technical Staff Member in the Systems Networking Department at the IBM T. J. Watson Research Center. He received his B.S. and M.S. degrees, both in electrical engineering, from the University of Notre Dame in 1990 and 1992, respectively. In 1996, after receiving his Ph.D. degree in computer science, also from Notre Dame, he joined IBM's Research Division as an Advisory Software Engineer. His primary professional focus is on system software for networking. He has contributed to multiple technical publications, patents and IBM products and has been named an IBM Master Inventor. He a member of the ACM and IEEE.

Renato Recio *IBM Systems and Technology Group, Austin, TX 78758 USA (recio@us.ibm.com).* Mr. Recio is IBM Fellow and System Networking CTO, responsible for IBM's Software defined Networking strategy and roadmap. For the past 15 years, he has played a leadership role in the strategy, architecture and design of future IBM system IO and Networks. His recent contributions include: Ethernet Virtual Bridging (EVB), Distributed Overlay Virtual Ethernet (DOVE) networks, EVB/DOVE adapter offload, and SDN service chaining. He has been a founding engineer and author of several input/output and network industry standards, including InfiniBand** (cluster network), IETF Remote Direct Data Placement (over TCP/IP), PCI IO Virtualization, Convergence Enhanced Ethernet (CEE), RoCE (RDMA over CEE), Fibre Channel over CEE, and IEEE 802.1Qbg EVB. He has filed over 200 patents, of which more than 100 have already issued. He has published dozens of refereed technical conference (e.g., IEEE and ACM) papers. He is an IEEE member and chaired the IEEE Data Center Converged and Virtual Ethernet Switching (DC CAVES) workshops.