SDN Is DevOps for Networking

ROB SHERWOOD

Rob serves as the CTO for Big Switch Networks, where he spends his time internally leading software architecture and externally evangelizing SDN to customers and partners. Rob is an active contributor to open source projects such as Switch Light and Floodlight as well as the Open Compute Project. He was the former chair of the ONF's Architecture and Framework Working Group as well as vice-chair of the ONF's Testing and Interoperability Working Group. Rob prototyped the first OpenFlow-based network hypervisor, the FlowVisor, allowing production and experimental traffic to safely co-exist on the same physical network and is involved in various standards efforts and partner and customer engagements. Rob holds a PhD in computer science from the University of Maryland, College Park.
rob.sherwood@bigswitch.com

Caught in a perfect storm of technology trends—including public and private cloud, Bring-Your-Own-Device (BYOD), converged storage, and VoIP—computer network management is reaching unprecedented levels of complexity. However, unlike server administrators whose tools have evolved with the times, network administrators are stuck using 20+-year-old box-by-box management tools. A new technology trend, Software-Defined Networking (SDN), promises to simplify network management. Although it seems like every vendor has its own definition of SDN, in this article, I make the claim that SDN is to networking what the DevOps movement is to server management: a way of making systems management easier to manage by adding programmable APIs that enable better automation, centralization, and debugging. In this article, I try to provide background on SDN, to snapshot its current and highly fluid state, and end with some predictions for what to expect next.

Networking Needs a "DevOps"

All types of networks, including campus, datacenter, branch office, wide area, and access networks, are growing at unprecedented speeds. More people with more devices are coming online and are accessing an increasing plethora of data. Although this is good for society at large, the reality is that networks themselves are becoming increasingly difficult for "mere mortals" to manage. In the past, sensitive data would exist on a single, dedicated, physical server in a fixed location with a clear physical DMZ policy "choke point" as the divide between the trusted and untrusted parts of the network. Today, sensitive data can be spread across multiple databases potentially distributed throughout the cloud on virtual machines that change physical location depending on load; thus, the single policy "choke-point" is a thing of the past.

While low-level packet forwarding devices have made amazing advances with speeds moving from 100 Mb/s server ports to 10 Gb/s and beyond, the management tools needed to operate and debug these devices have stagnated. Indeed, operators of today use the same basic command-line syntax and tools to configure routers as when I first started administering networks 20 years ago. The only thing that seems to have changed is that we used to telnet to these boxes but now we use SSH! Furthermore, network administrators are caught between a rock and a hard place because the management interfaces for the network devices are typically closed, vertically integrated systems that resist enhancement or replacement.

Despite going through the same growing pains, server administrators dodged these problems with a variety of automation and centralization tools that can roughly be grouped under the term "DevOps." DevOps infuses traditional server administration with best practices from software engineering, including abstraction, automation, centralization, release management, and testing. The server ecosystem is quite different from networking because it has many open interfaces: server admins can supplement, enhance, or replace software components of their systems, including configuration files, whole applications, device drivers, libraries, or even the entire operating system if desired. As a result, server administrators

were able to manage growing server complexity by replacing and automating critical components of their management stack with tools such as Puppet [1], Chef [2], and others. In other words, although server administrators have the same problems in terms of scale and complexity as network administrators, they were able to solve their problem with DevOps-style deployments because the server ecosystem has open and programmable interfaces.

SDN Promises an Interface to Unlock Networking

Centralization, automation, and better debugging sound like good goals, but the closed and vertically integrated nature of most switches and routers makes it unclear how to apply them to networking. SDN promises to create an application programming interface (API) for networking and thus unlock DevOps' same desirable properties of automation, centralization, and testing.

The term SDN was first coined in an *MIT Technology Review* article [3] by comparing the shift in networking to the shift in radio technology with the advance from software defined radios. However, the term is perhaps misleading because almost all networking devices contain a mix of hardware and software components. This ambiguity was leveraged and exacerbated by a litany of companies trying to re-brand products under the "SDN" umbrella and effectively join the SDN bandwagon. As a result, much of the technical merit of SDN has been lost in the noise.

The SDN movement, originating from Stanford University circa 2007, was first exemplified by the OpenFlow protocol. OpenFlow is an open protocol and is currently maintained by the vendor-neutral Open Networking Foundation [4]. OpenFlow exposes a remote API for managing the low-level packet forwarding portions of network devices, including switches, routers, access points, and the like. At a high level, OpenFlow abstracts packet forwarding devices as a series of "match action" tables. That is, when a packet arrives at a device, it is processed through a series of prioritized lookup tables of the form "if the packet matches *MATCH,* then apply *LIST OF ACTIONS,*" where the list of actions can be anything from "send packet out port X," "decrement TTL," or rewrite one of the packet header fields. By creating this abstraction layer and interface, network admins can, in a programmable way, manage the forwarding rules of their networks in an automated and centralized manner.

Many-Layered APIs of SDN

From the first successes of OpenFlow [5], SDN began to expand and consider new use cases and deployments. As with any vibrant software ecosystem, many APIs—both complementary and competing—have begun to emerge. In addition to APIs like OpenFlow for managing packet forwarding logic, interfaces for managing configuration, tunneling, as well as more traditional

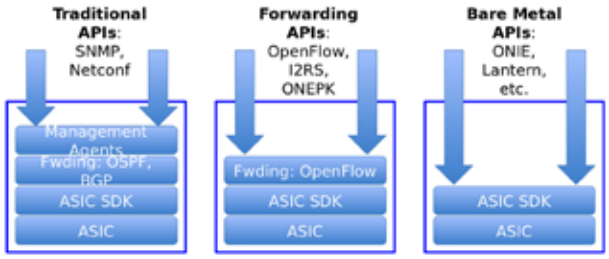

Figure 1: SDN has many layers, from the traditional APIs, to the forwarding APIs that were the first target of SDN, to the bare metal APIs.

APIs for statistics monitoring and debugging are being viewed as SDN. Most recently, much like with servers, the lowest-level "bare metal" hardware APIs are being exposed, allowing enterprising startup companies and DIY types to write their own network operating systems from the ground up. In other words, SDN is bringing the networking ecosystem closer to the server DevOps ecosystem where an administrator can choose the right API/tool for the task and automate and centralize common tasks.

As with any complex and rapidly evolving system, tracking all of the APIs, protocols, ideas, and works-in-progress is impractical, but here I try to provide a hopefully representative snapshot of the state of SDN. Figure 1 provides a visible map of some of the existing layers.

Forwarding Plane APIs

Probably the most important and novel aspect of SDN is the ability to programmatically manage low-level packet forwarding. OpenFlow itself has evolved quite a bit since its debut 1.0 release in 2009. More modern versions of OpenFlow have added support for richer packet actions (e.g., NAT, tunneling, metering), more extensible matches, multiple tables, IPv6 support, and even batched "bundled" commands with the most recent version: OpenFlow 1.4.0 [6]. In addition to new forwarding capabilities, the Open Networking Foundation (ONF) is exploring better abstractions for wired forwarding hardware as well as for optical and wireless technologies. Rather than replacing traditional routing forwarding decisions, the IETF's Interface to the Routing System (I2RS [7]) seeks to provide an API for merging programmatic packet forwarding with packet forwarding decisions inferred from traditional routing protocols like BGP, IS-IS, OSPF, and others. Not to be left out, traditional networking vendors have created their own forwarding APIs, including Cisco's onePK and Juniper's Junos Space.

Configuration and Statistics APIs

Besides low-level packet forwarding, networking devices have a dizzying array of tunable configuration parameters and statistics. Many protocols, such as SNMP and Netconf that long predated OpenFlow exposed APIs ("MIBs" in SNMP, "schemas"

in Netconf), allow network admins to tweak configuration settings and monitor statistics like port counters. Newer APIs, like ONF's OpenFlow Config protocol [8] and Open vSwitch's DB management API [9], supplement existing APIs by adding support for managing tunnels and virtual switches (i.e., by adding and removing virtual ports). Additionally, many of these APIs have support to enable and configure packet sampling protocols like NetFlow and sFlow, which are critical for in-depth traffic analysis.

Bare Metal and Open Hardware APIs

Whereas the above APIs build on top of existing vendor software, it is increasingly possible to write directly to the low-level hardware APIs and replace vendor software altogether. By comparison, if writing packet forwarding rules is like writing your own application, then writing to the bare metal hardware is like writing your own operating system. Although writing the entire network stack is not for the faint-of-heart, it can be necessary to overcome limitations of existing vendor stacks or to accomplish something completely revolutionary. Writing to the bare metal is made possible by two recent changes in the ecosystem: a standardized network device boot loader and open ASIC APIs.

The Open Network Install Environment (ONIE [10]) is an open source boot loader available for an increasing number of networking devices, particularly datacenter switches. In server terms, ONIE provides functionality that is one part PC BIOS and one part grub/lilo/sysimage. A network admin can use ONIE to add/remove/reset the switch operating system over the network. In other words, using ONIE, it is possible to network boot (or even dual boot!) an arbitrary network operating system on to an ONIE-enabled network device. Think of it as PXE for switches and routers. ONIE is hosted and sponsored by the Open Compute Project (OCP [11]), which, among other aspects, includes open hardware designs and specifications for networking devices.

To achieve high speeds, modern networking typically requires special purpose hardware, such as an Application-Specific Integrated Circuit (ASIC). Historically, the APIs to program these ASICs have been closed and access to them tightly controlled via strict non-disclosure agreements. However, more recently, ASIC manufacturers are moving to the "bare metal" bandwagon and have started to make the APIs public. For example, ASIC manufacturers Centec and Mellanox have begun to publish their APIs in their Lantern [12] and Open Ethernet [13] projects, respectively. Other ASIC manufacturers seem likely to follow suit, so this trend seems likely to increase over time.

Impact from Market Forces

Although SDN is primarily a technology movement, it would be an error to assume that its traction is purely a result of a superior architecture. As technologists, we like to ignore the economics, but history is filled with technologies that didn't succeed despite superior design. In particular, technologies similar to SDN have come and gone in the past without comparable traction, including IETF's ForCES [14] and the field of active networking. So a critical question is, why is SDN achieving industry traction where similar technologies have not?

The answer is that the underlying market forces of networking have significantly changed. Large datacenters mean that more money is being spent on networking than ever before, which encourages both more competition as well as bigger gains from commodities of scale. Historically, packet forwarding ASICs were only created by pure networking vendors for inclusion into their own vertically integrated products. As a result, the market rewarded vertically integrated closed systems because that best protected the companies' ASIC investments. But, with the rise of large datacenters, sufficient ASICs were being sold that highly specialized "ASIC only" companies became commercially viable. Soon, companies like Broadcom, Marvell, Fulcrum, and others began to create switching ASICs and sell them to others without owning the full solution. Competition for this new commodity "merchant" silicon space increased, and now we are beginning to see strong market forces come into play in terms of lower costs and additional features. This is all because merchant silicon companies have the incentive to sell more and better ASICs—not more and better boxes. It is a result of this competition that these same companies are breaking industry norms and publishing their APIs—and thus enabling SDN.

Another effect of large datacenters is the convergence of compute, storage, and networking. Administrators are increasingly buying these resources as integrated solutions, and vendors are reacting in turn. The result is that traditional networking companies are starting to sell products that integrate compute and storage (e.g., Cisco's UCS product), and traditional computer companies are starting to acquire networking companies (e.g., HP bought 3Com, Dell bought Force10, IBM bought Blade Networks). The once very stable networking market is full of new and significant competition, with each company looking for ways to differentiate itself from the rest—including opening up networking devices by implementing SDN protocols like OpenFlow.

Conclusions and Predictions

Networking administrators are adopting SDN for many of the same reasons that server administrators adopted DevOps: automation, centralization, and ease of debugging. Historically, network devices have been vertically integrated closed software stacks with few mechanisms to replace or extend their functionality. However, recent changes in the market are causing vendors to shift their business models and open up their devices to programmable access through a suite of APIs. The result appears to be a trend towards a more extensible and vibrant third-party software-driven networking ecosystem.

Perhaps more interesting than any one specific API are the applications that are enabled by using combinations of APIs. For example, imagine an application that makes API calls to all of the devices in the network to set up sFlow sessions, monitor the dynamically changing traffic, and then make further API calls to readjust traffic engineering policies via OpenFlow. Such combinations will allow networks to be more easily managed and scale up to the demands from BYOD, VoIP, and future technology trends.

In terms of predictions, my big claim is this: after 20+ years of closed software stacks in networking devices, the genie is out of the bottle. I believe that as in the transition from the IBM mainframe to the PC or from closed cell phones to modern, open API smartphones, we will see networking go through a renaissance. We will see switch operating systems and applications that are entirely open source, and applications that do more niche and specialized tasks. We will see the cost of hardware drop significantly: just to put a number to it, I believe we will see the cost of 10G Ethernet switches drop below $75 per port before 2015. This explosion of new ideas, lower cost hardware, and innovative networking features will change how networking consumers view their networks. In other words, I believe that with an open network, operators will be empowered to create and deploy innovative new features that will change networking from a cost center into a new source of revenue in terms of novel products for their customers. The really fun question becomes, what will be the killer app that no one thought of until everyone needed it?

References

[1] Puppet: http://www.puppetlabs.com.

[2] Chef: http://www.getchef.com.

[3] Kate Greene, "10 Breakthrough Technologies: TR10—Software-Defined Networking," *MIT Technology Review*, March/April 2009: http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/.

[4] Open Networking Foundation: https://www.opennetworking.org.

[5] Open Networking Foundation, Solution Briefs: https://www.opennetworking.org/sdn-resources/sdn-library/solution-briefs.

[6] Open Networking Foundation, OpenFlow 1.4.0 Wire Protocol Specification: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[7] I2RS: https://datatracker.ietf.org/wg/i2rs/charter/.

[8] OpenFlow Config Protocol: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow-config.

[9] Open vSwitch's DB management API: http://www.openvswitch.org.

[10] Open Network Install Environment (ONIE): http://onie.github.io/onie/docs/overview/index.html.

[11] Open Compute Project: Networking: http://www.opencompute.org/projects/networking/.

[12] Centec Lantern ASIC APIs: http://www.centecnetworks.com/en/OpenSourceList.asp?ID=260.

[13] Mellanox Open Ethernet Project: http://www.mellanox.com/openethernet/.

[14] IETF's ForCES: http://datatracker.ietf.org/wg/forces/charter/.