Faucet
Deploying SDN in the Enterprise

JOSH BAILEY AND STEPHEN STUART

USING OPENFLOW AND DEVOPS FOR RAPID DEVELOPMENT

The 2008 publication of "OpenFlow: Enabling Innovation in Campus Networks" introduced the idea that networks (originally campus and enterprise networks) can be treated more like flexible software rather than inflexible infrastructure, allowing new network services and bug fixes to be rapidly and safely deployed.[7]

Since then many have shared their experiences using SDN (software-defined networking) and OpenFlow in wide area and data center networks, including at Google.[10] This article returns to enterprise and campus networks, presenting an open-source SDN controller for

such networks: Faucet. The Faucet controller provides a "drop-in" replacement for one of the most basic network elements—a switch—and was created to easily bring the benefits of SDN to today's typical enterprise network.[5]

SDN enables such safe and rapid development and deployment of network features through automated testing of both hardware and software, without time-consuming manual lab testing. As described here, a complete control-plane upgrade can be done, while the network is running, in a fraction of a second.

Security of networks is a concern for all network operators and users. A zero-day attack on the network itself is especially worrisome because it can impact the security of all users and services on the network. Therefore, it is critical that network operators have a way of responding rapidly, both to deploy new security features or mitigate vulnerabilities in advance of an attack and to restore a network currently undergoing an attack as quickly and completely as possible, with as little risk as possible. SDN builds this ability to change and respond into the network itself at a very low level, which is beyond the reach of an external security device such as a firewall.

Faucet has been tested, and it performs. It has been deployed in various settings, including the Open Networking Foundation, which runs an instance of Faucet as its office network. Faucet delivers high forwarding performance using switch hardware, while enabling operators to add features to their networks and deploy them quickly, in many cases without needing to change (or even reboot) hardware. Furthermore, it interoperates with neighboring non-SDN network devices.

Faucet was built on the OpenFlow 1.3 standard.[8] Without the availability of commercial hardware supporting this standard, it would not have been possible. Multiple vendors now ship hardware that supports OpenFlow 1.3, specifically with support for multiple flow tables and IPv6. To minimize vendor-specific logic in the controller, vendors were encouraged to support key features in the OpenFlow 1.3 standard in a consistent way. This reduced initial development and support cost, and it simplified bug reporting and automated testing.

While SDN as a technology continues to evolve and become even more programmable (e.g., with the P4 programming language), Faucet and OpenFlow 1.3 hardware together are sufficient to realize benefits today. This article describes specifically how to take advantage of DevOps practices ("push on green"[6]) to develop and deploy features rapidly. It also describes several practical deployment scenarios, including firewalling and network function virtualization.

MANAGEMENT OF ENTERPRISE NETWORKS TODAY

Many enterprise networks consist of multiple layers of switches, often with VLANs to partition users into different administrative domains (for example, sales separated from engineering). Connected to these switches is a diverse range of appliances and devices[9] required to manage the network and implement security policy, sometimes requiring complex and fragile forwarding policies to put them in the path of packets. Non-SDN switches are not programmable (by definition), so their forwarding and security policy is defined by what each

vendor's proprietary configuration language provides. In some cases, an external system, such as an IDS (intrusion detection system), can make coarse changes to a network to implement dynamic security policy (e.g., disable a host port if the IDS determines that a host on that port has been infected with malware).

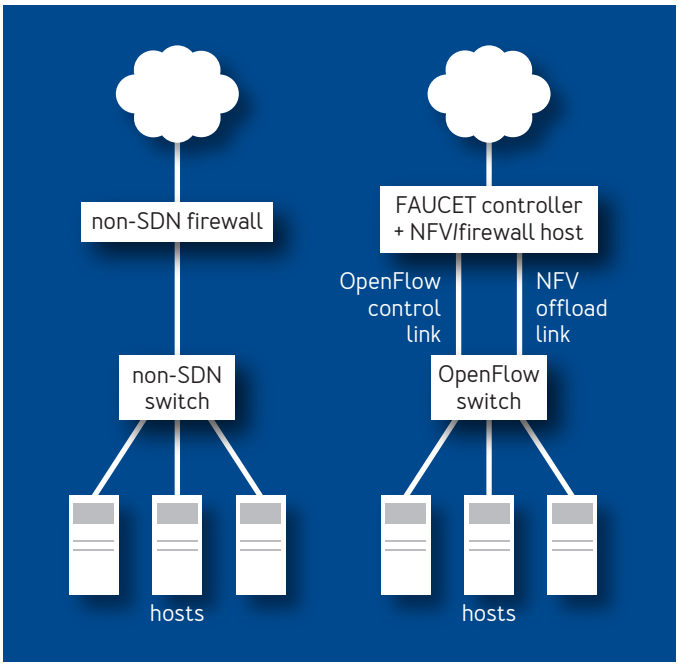
Today's network operators are responsible for administering and integrating that wide range of appliances and devices, and it may be difficult to implement a specific security policy if the available devices do not have the necessary features to achieve the desired policy result. To operate and maintain a secure network with inflexible tools requires considerable skill and effort, and since the network cannot be programmed, opportunities to automate are limited. This is especially true when vendors either do not provide programmability or provide only proprietary automation technologies that operate best on a particular vendor's equipment.

DEPLOYING FAUCET IN TODAY'S ENTERPRISE NETWORKS

To realize the benefit of SDN you have to be able to deploy it. Deployment has to be easy and, ideally, incremental. To fit these real life requirements, Faucet was designed to replace a conventional non-SDN switch, one for one, as shown in figure 1, realizing the benefits of SDN in that network without necessitating notable infrastructure changes.

Faucet is deployed as a unit of two systems: a controller host (typically a host running Ubuntu Linux, running the Faucet controller application) and an OpenFlow switch (e.g., an Allied Telesis x930 series switch), which are

FIGURE 1: **NON-SDN AND FAUCET SDN COMPARISON**



directly connected. The controller takes one configuration file, which describes which ports are in which VLANs (or if a port is in a VLAN trunk). The entire installation process, including creation of the configuration file (which resembles a non-SDN switch configuration file), has been reported to take only minutes in recent deployments.

Stopping at simple deployment, however, won't realize meaningful benefits. In fact, replacing a single switch with a controller machine (note that one controller machine can control many switches) *and* a switch takes more space and power. The benefits come from having the controller and control software both separate from the switch hardware,

and entirely within the network operator's control, rather than being a closed vendor provided appliance that cannot be reprogrammed.

Controllers can be numerous for high availability—Faucet supports redundant controllers—and as big or as small as required. Faucet is run in production at one site using a Raspberry Pi as a controller, which is practical because with Faucet the switch hardware does the forwarding—the controller does not have to be very powerful, because it does not have to forward traffic, leaving that to the higher-performance switch hardware.

PUSH CODE TO THE NETWORK ON GREEN

"Push on green" refers to a philosophy of being able to create and test software in an automated fashion such that it is easy to detect a "green" condition of code—ready to roll out with as few bugs as possible.[6] Google has published some of its strategies for deploying and managing large reliable software systems in the recent book, *Site Reliability Engineering*.[3] SDN promises to help apply these strategies to networking software.

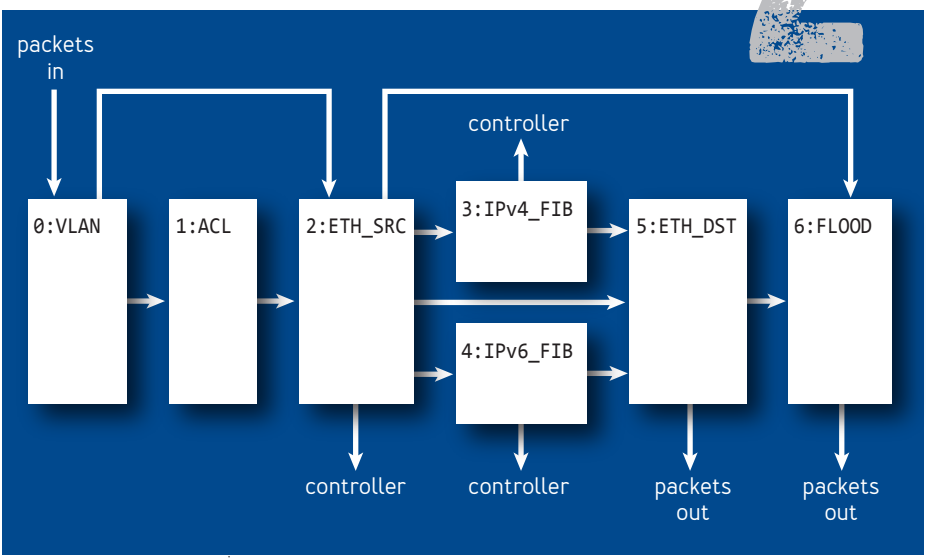
In keeping with this, the Faucet software stack includes a unit-testing framework,[2] so that new features can be developed with unit tests, and tests can be run against both simulated (Mininet virtual network) and hardware switches. Tests detect "green," and the operator can "push on green" with confidence that the system will work as tested. This accomplishes both feature-level and system-level testing—it is possible to catch system and integration problems at development time, well before deployment or even lab testing.

As an example, many non-SDN switches implement unicast flooding as part of the learning process,[4] so that the switch can discover which hosts are connected to which ports. Learning works in this way: a host sends an Ethernet packet with a destination unknown to the switch, and the switch floods the packet to all ports in the hope that the intended destination host will respond. This may not be desirable for security reasons, and in many non-SDN switches this behavior is hard-coded.

A Faucet feature that implements switch learning exclusively from ARP (Address Resolution Protocol) and neighbor discovery packets relieves Faucet from the need to flood unicast packets. The feature was implemented, tested with unit tests that included hardware and software, and pushed to github. On the controller machine, the operator ran "git pull" and then "service faucet restart," accomplishing a complete controller upgrade and restart. Forwarding was interrupted for less than one second. Other features involving changes to the controller, even more ambitious features, can be deployed the same way.

As shown in figure 2, Faucet implements all its features using OpenFlow 1.3 and multiple tables, and today implements VLAN switching, IPv4 and IPv6 routing (both static and via the BGP routing protocol), ACLs (access control lists), port mirroring, and policy-based forwarding. The switch does all the forwarding, and no "hybrid" mode functionality is used on the switch. Hybrid mode is where a switch uses a mixture of nonprogrammable, non-SDN local processing, together with OpenFlow control. Faucet does not need such switch-local processing, and in our experience hybrid mode increases complexity and limits

FIGURE 2: **FAUCET'S ALL OPENFLOW PIPELINE**



programmability by introducing the possibility of conflict between local and OpenFlow control.

A tiny fraction of traffic is copied to the controller so it can learn which hosts are on which ports and so the controller itself can resolve next hops (if routing is configured). The controller is generally idle unless hosts are added or move between ports (in which case the controller reprograms the pipeline as appropriate). Faucet has basic protection against control-plane attacks (for example, limiting spoofed Ethernet MAC addresses). Because the pipeline is entirely programmed by the controller, the network operator is free to make arbitrary changes to forwarding behavior by changing the controller software.

When the hardware switch boots, it establishes an

OpenFlow connection to the controller. The controller provisions the initial pipeline, including expectations for VLAN tags and any ACLs if configured, and adds "default-deny" rules (all unknown traffic is explicitly dropped). When a new host is detected, the switch sends a copy of the Ethernet header to the controller and (if unicast flooding is enabled) floods it to all other ports on the same VLAN or (if unicast flooding is disabled) floods it only if the packet is an IPv6 neighbor discovery or ARP packet. The controller then programs flows to cause future packets from this Ethernet source address to be forwarded by the switch. These flows periodically time out and are refreshed by the controller as necessary, which allows the switch to conserve resources.

ROLE OF THE CPN AND SECURITY OF THE CONTROL PLANE

The CPN (control-plane network) connects the controller machine and the switch on a dedicated port. In many deployments this is simply a good-quality three-foot Ethernet cable, which has been observed in production to be no less reliable than the internal connection between the CPU and data plane in a non-SDN switch. Indeed, should that three-foot cable fail, it can easily be replaced. In larger deployments where one controller machine controls several OpenFlow switches, the three-foot cable can be replaced by several Ethernet cables and a good-quality non-SDN switch. The controller connection between the switch and the Faucet controller can be secured with certificates or even MACsec-capable interfaces. (MACsec is the IEEE 802.1AE MAC security standard.)

Many switches allow configuration of the handling

of the control connection being lost: "fail secure" (keep forwarding and using currently programmed flows until they expire) or "fail standalone" (revert to being essentially a nonprogrammable switch). Faucet implements expiry times on all flows, which causes forwarding to cease if no controller can be reached for a configurable period, so Faucet expects the switch to be in "fail secure" mode. It is possible to replace the CPN (and switch), and this has been done in production without interrupting forwarding if done within the flow expiry time. (It is generally not possible, on the other hand, to replace the back plane in a standalone non-SDN switch without interrupting forwarding.)

POWERFUL CONTROLLERS ARE OPPORTUNITIES

In a non-SDN switch the embedded CPU is generally power- and cost-optimized. With Faucet, however, the controller can be a general-purpose computer or, indeed, a powerful server-class computer. This represents an opportunity to use the controller machine hardware as an open-source network coprocessor or an alternative to a firewall appliance—the Faucet switch effectively provides lots of extra ports to a powerful machine.

In the first author's own deployment[1] the controller machine also runs Ubuntu's ufw (uncomplicated firewall) package, which implements NAT (network address translation) and firewalling; Bro (https://www.bro.org/), which is an IDS (intrusion detection system); and Internet Software Consortium's dhcpd (Dynamic Host Configuration Protocol daemon) to assign addresses dynamically. All three VLANs are trunked to a dedicated port on the controller machine. Faucet has ACLs on the host ports,

which prevent hosts from spoofing the controller's IP addresses in each VLAN (so that proxy ARP attacks are not possible). Faucet ACLs operate across layers, so it is possible for an ACL entry to match, for example, Ethernet type, as well as IP address and MAC address.

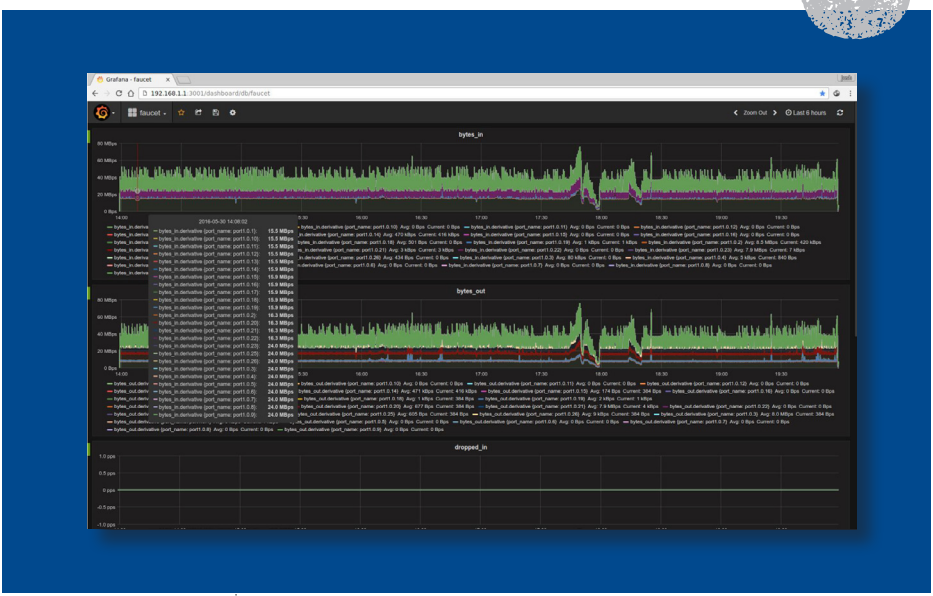
More complex configurations are possible; for example, it would be possible, via the controller's trunk port, to assign a Linux container to each VLAN and run a separate iptables chain for every switch port. This achieves complete isolation and complete security policy customization on a port-by-port basis, without requiring any changes to the switch.

Faucet departs from common current network management practice in that it does not implement SNMP (Simple Network Management Protocol). Instead, the Faucet controller pushes statistics (bytes, packets, in and out from each port) to an external system. Faucet supports InfluxDB (https://influxdata.com/), which is an open-source time series database. Using Faucet together with an open-source visualization system, Grafana (http://grafana.org/), it is possible to construct dashboards and run queries on current and historical data, as shown in figure 3. Faucet can also produce a JSON (JavaScript Object Notation) log of statistics that could be translated and input into another system.

NORTHBOUND API

The SDN community continues to debate APIs that business, security, or other applications external to the controller should use to control the controller (for example, to ask the controller to prioritize a given user's

FIGURE 3: **USING FAUCET AND GRAFANA TO CONSTRUCT DASHBOARDS AND RUN QUERIES**



traffic on the network). Faucet does not have a "one true" northbound API, because a generic API is not required. An operator can develop an application on top of Faucet that delivers Faucet a new configuration file and asks Faucet to apply it (e.g. to change a user VLAN). Or, an operator might directly modify the controller code to add the desired feature, or add some other API convenient to the operator's needs (for example, to integrate the Bro IDS).

RELATIONSHIP TO OTHER SDN CONTROLLER PROJECTS

The SDN community has several controller projects. Two well-known ones are ODL (OpenDaylight, https://www.opendaylight.org/) and ONOS (Open Network Operating

System, http://onosproject.org/). Both controllers have many ambitions and have already served as technology demonstrators. As operational experience with SDN in the wider industry is still in its early stages, however, it is important to provide low-risk, incremental migration paths between today's networks and those aimed for by ODL and ONOS. Faucet could provide one such path and help inform the ongoing design of new network abstractions and programming frameworks. Furthermore, today's network operator community has traditionally not written its own software. While DevOps may be mainstream for many service operators, it is not for network operators, and it is important to make the benefits of such practices directly relevant to them.

CONCLUSION

The benefits of SDN have been difficult to realize because of a lack of software that is accessible to today's network operator community. While still a very simple system, Faucet could be useful enough to operators that they may take the next step toward migrating to SDN, enabling them to adopt and enjoy the specific benefits of the rapid feature development, deployment, and testing Faucet provides.

Downloading faucet

Faucet's source code for development can be found at https://github.com/REANNZ/faucet, or can be downloaded at https://pypi.python.org/pypi/ryu-faucet/ (including Docker images) for easiest installation.

References

1. Bailey, J. 2016. NFV/firewall offload with Faucet. Faucet SDN; http://faucet-sdn.blogspot.co.nz/2016/05/nfvfirewall-offload-with-faucet.html.
2. Bailey, J. 2016. Unit-testing framework. Faucet SDN; http://faucet-sdn.blogspot.co.nz/2016/06/unittesting-hardware.html.
3. Beyer, B., Jones, C., Petoff, J., Murphy, N. R., eds. 2016. *Site Reliability Engineering.* O'Reilly Media.
4. Cisco. 2016. Unicast flooding in switched campus networks; http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6000-series-switches/23563-143.html.
5. Faucet; https://github.com/REANNZ/faucet.
6. Klein, D. V., Betser, D. M., Monroe, M. G. 2014. Making "push on green" a reality: issues and actions involved in maintaining a production service. Research at Google; http://research.google.com/pubs/pub42576.html.
7. McKeown, N., et al. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38(2): 69-74.
8. Open Networking Foundation. 2013. OpenFlow Switch Specification, version 1.3.3; https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.3.pdf.
9. Sherry, J., Ratnasamy, S. 2012. A survey of enterprise middlebox deployments. Technical Report UCB/EECS-2012-24. University of California at Berkeley; http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-24.pdf.

10. Vahdat, A. 2015. Pulling back the curtain on Google's network infrastructure. Google Research Blog; http://googleresearch.blogspot.com/2015/08/pulling-back-curtain-on-googles-network.html.

Related articles

The Road to SDN
An intellectual history of programmable networks
Nick Feamster, et al.
http://queue.acm.org/detail.cfm?id=2560327

OpenFlow: A Radical New Idea in Networking
An open standard that enables software-defined networking
Thomas A. Limoncelli
http://queue.acm.org/detail.cfm?id=2305856

A Purpose-built Global Network: Google's Move to SDN
A discussion with Amin Vahdat, David Clark, and Jennifer Rexford
http://queue.acm.org/detail.cfm?id=2856460

JOSH BAILEY *is a staff software engineer at Google, working on network management and research projects for the past 11 years. He is based in Wellington, New Zealand.*

STEPHEN STUART *is a distinguished software engineer at Google for the past 13 years, based in Mountain View, California, USA.*