ClosedFlow: OpenFlow-like Control over Proprietary Devices

Ryan Hand
University of Colorado, Boulder
ryan.hand@colorado.edu

Eric Keller
University of Colorado, Boulder
eric.keller@colorado.edu

ABSTRACT

Software Defined Networking (SDN) offers unprecedented control to network administrators. With datacenters, because of the rapid build-out (new equipment installed) and the pervasiveness of virtualization (software switches are widely used), SDN is rapidly gaining traction. The reality for enterprises is that there is already an existing (legacy) network and, for the most part, companies don't have the excess capital to throw away their current investment and replace all of their network equipment with SDN capable devices. This means that uptake of SDN in enterprises is understandably slow. What is needed is a way for companies to gradually transition to SDN. In this paper, we present ClosedFlow, a system which incorporates techniques for exercising SDN control over existing proprietary hardware which closely mimics the fine grain control available in OpenFlow. This allows enterprises to control the network through a centralized controller, taking advantage of SDN's benefits today with no new investment, and gradually transition the hardware to SDN enabled hardware (*e.g.,* OpenFlow) over time as part of their typical equipment replacement lifecycle.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Network operating systems*

General Terms

Design, Experimentation, Management

Keywords

software-defined networking; legacy devices; OpenFlow

1. INTRODUCTION

SDN substantially lowers the barrier for adding new networked services, such as flexible access control [6], Web server load balancing [12, 23], energy-efficient networking [13], adaptive network monitoring [14], and seamless user mobility and virtual-machine migration [9]. A software-defined network runs these services on a logically-centralized controller that uses a standard API (such as OpenFlow [17]) to install packet-handling rules in the underlying switches. In addition, SDN naturally supports network virtualization, where each virtual topology runs its own controller applications tailored to the needs of a particular "tenant" or class of traffic [20, 1, 3]. SDN can also enhance network scalability and reliability by leveraging advances in distributed systems to manage network state separately from any application-specific control logic [15]. Finally, and potentially most importantly, SDN can greatly enhance network security through the ability to dynamically adapt to changing threats [21, 11].

Despite the many advantages and possibilities of SDN, enterprises are understandably slow to adopt the new technology. For network administrators that want to move to a software-defined network today, the main option is effectively to "fork lift" their existing infrastructure to SDN-capable devices (illustrated in the top half of Figure 1)[1]. Once they fork-lift replace their hardware, they can gradually transition over to SDN control with 'ships-in-the-night' behavior supported in most commercially available SDN switches, where traffic can be designated to be handled by either by the SDN-capable portion or by legacy protocols. In one attempt at an alternate transition, Panopticon effectively proposes sprinkling a few SDN switches (*e.g.,* at the edge) and configuring the legacy switches to act as tunnels between the SDN switches [16]. While this allows SDN functionality to be incorporated, it does require new hardware along with specialized configuration on the legacy devices to handle operating in support of SDN enabled switches.

In this paper we propose a different transition with ClosedFlow, illustrated in the bottom half of Figure 1, which allows SDN control over existing legacy hardware. That is, with no new hardware purchases an SDN controller, such as Floodlight [5] or Ryu [4], can run network controlling applications but instead of targeting OpenFlow switches, legacy devices can be targeted (with no modifications to the applications, only to the code that interfaces with hardware).

While many have been 'pushing configs' from a centralized server for years to realize SDN-like control over legacy switches [10, 7], the key question is how closely we can mimic the current leading embodiment of SDN (OpenFlow) with the proprietary configuration interfaces of legacy devices. While SDN is generally considered more general than OpenFlow, we focus on OpenFlow because it represents a well-defined and open interface designed by the com-

[1]Some vendors do offer firmware updates to some more recent devices, but there are many devices still being used in production networks which are either from vendors that have yet to support OpenFlow or which are older than the models the vendor provides an OpenFlow firmware update for.

HotSDN'14, August 22, 2014, Chicago, IL, USA.

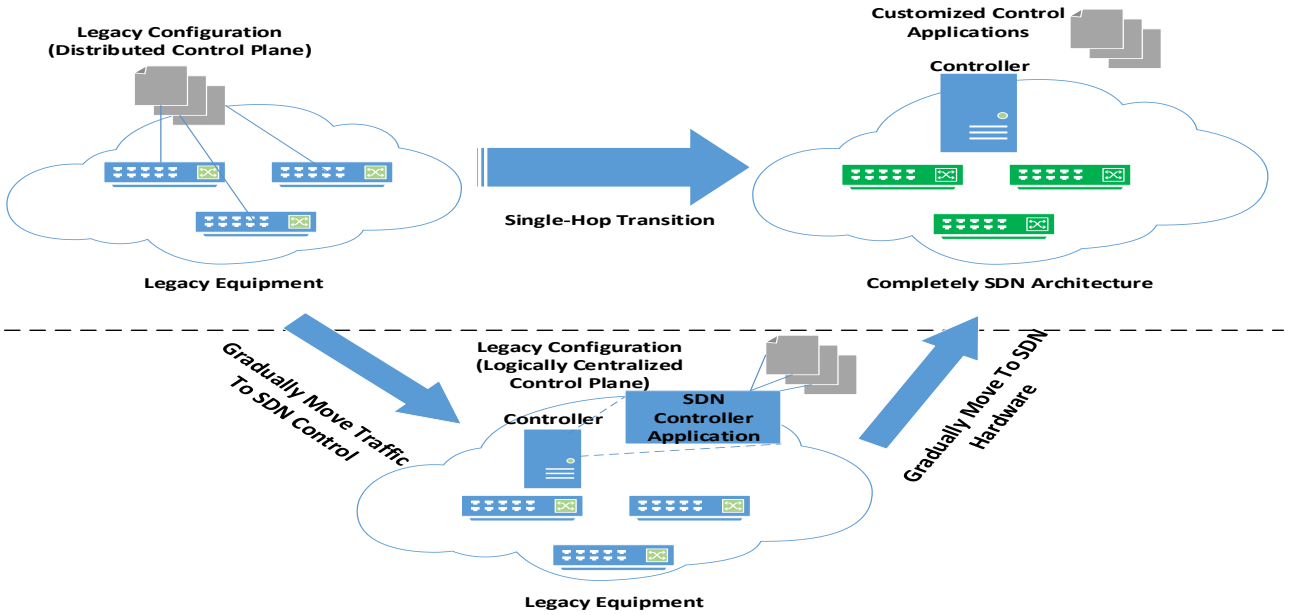

Figure 1: Single Phase SDN Transition vs. Multi-Phased Configuration and Hardware Transition

munity, and as such targeting it represents an interface that is undeniably SDN. We make the following contributions.

- We explore and answer this key question by demonstrating the four main capabilities in OpenFlow: establishing a control channel, automatically discovering the topology for a network-wide view, modifying a flow table with entries that specify matching packet headers and actions to forward or drop packets, and handling the 'send-to-controller' actions. In Section 2, we demonstrate these four capabilities, though handling the 'send-to-controller' action requires a compromise. Our prototype *static flow pusher* application is discussed in Section 3.

- We evaluate the system with our prototype targeting 10 year old Cisco switches. We show that while the table sizes of legacy switches is not as large as modern SDN switches, the legacy switches have flow table optimization technology which allows for many flow table entries to be set before processing needs to be relegated to lookups in slower memory such as DRAM (discussed in Section 4).

- We illustrate that if we don't limit ourselves to OpenFlow, we can enable much more capabilities by tapping into functionality that has been present in commercial switches for over a decade (discussed in Section 5).

2. REALIZING CLOSEDFLOW

The key question is whether we can provide an OpenFlow interface (from the SDN controller's perspective) to control a network of legacy switches. That is, we are not just proposing a centralized interface to legacy switches, with vendor specific configurations. We are proposing that the control associated with OpenFlow (and capitalized on in the many research works illustrating the benefits), can be achieved with legacy switches and with hardware performance.

To explore this, in each of the subsections we will illustrate the four main characteristics of an OpenFlow network: (i) A communication channel between a central controller and each switch, (ii) topology discovery, (iii) matching packets up to layer 4 and applying standard actions (*e.g.,* drop, forward), and (iv) handling the special action 'send-to-controller' with packet-in messages. Here, we focus on Cisco configuration for switches with a with a minimum IOS of 12.2(44)SE (which we have running on our over 10 year old Cisco 3550 switches), but we note that similar capabilities are present in all major vendor's switches.

2.1 Controller-switch Control Channel

A central requirement in SDN is that a centralized controller is able to communicate with each switch in the network and not need to be physically (directly) connected to each switch. Ethane overcame this through the use of spanning-tree protocol, commonly used to support plug-and-play Ethernet networks.

One challenge we face is that we are asking each switch in our topology to operate over layer-3 interfaces (in order to support matching at layer's 3 and 4). Because of this, we do not have layer-2 protocols, namely spanning-tree protocol, at our disposal to automatically discover and calculate paths between each switch and the controller (for control traffic).

To facilitate a communication channel and topology state awareness between switches and our controller, we chose to run a minimal instance of the Open-Shortest-Path-First (OSPF) routing protocol. To be clear, this instance is segregated from data flow traffic, which is forwarded based on route-map configurations and access-list match conditions.

Our OSPF instance includes advertisements for the Loopback management interfaces of each switch, a point-to-point connection between switches, and a VLAN dedicated for communication with the controller(s). With this, we enabled an in-band overlay control channel and remote access (SSH or telnet) to each switch for control traffic, such as pushing new flow rules.

New switch installation does require some minimum configuration. For our architecture, the below basic configurations are sufficient for the installation of a new switch.

- **Set IP address for interface Loopback 0**
- **Configure "routed" interfaces for switch-to-switch links**
- **Configure OSPF Instance and Set Router-ID to Loopback 0 IP**
- **Advertise Loopback and point-to-point networks in OSPF**
- **Set up remote access (*e.g.,* SSH or Telnet)**
- **Set Enable Mode Password**

This effectively establishes the line of communication with our controller, and enables remote access facilities to allow the controller to push configurations to each device.

2.2 Topology Discovery

A second requirement for an SDN network is for the controller to have a network-wide view, including a view of the entire topology. With Ethane, this was achieved by each switch periodically sending link-state information to the controller.

In our architecture, we could also follow Ethane's approach. The first is to use remote logging to the controller from each switch, to simply log adjacency changes. This would allow the controller to store topology state and maintain its control channel in the event of a link failure.

Alternatively, we can capitalize on the fact that in OSPF, each node in the network has complete visibility over the entire topology. This would require the controller to run the complete OSPF routing protocol and expose the current topology to the SDN controller. We believe the first approach is simpler and lighter-weight, though we have not implemented the second approach to say with certainty.

Our stored topology information will serve as an important reference point for the controller to know what route-map to apply to a specific interface. We also store ACL and route-map configuration information for each switch to keep track of existing match rules and route-map sequence numbers (explained further in Section 2.3).

2.3 Packet Matching and Applying Actions

The third requirement is that the SDN controller have the fine-grain ability to control how individual flows are handled, as opposed to tuning knobs on a variety of standard routing protocols. In an OpenFlow network, flow rules along with the forwarding behavior can be defined fairly simply.

In targeting legacy switches, we can achieve a similar level of control with a combination of access-control lists (ACLs), Route Maps, and Interface configuration.

- *Access control list* – ACLs specify the match conditions to classify the packet, and whether to permit or deny the traffic. If the OpenFlow action is to drop, that is set here. If the action is to forward out a specific port, the ACL will be configured to permit and a route map will handle.
- *Route Map* – Route maps are applied at an inbound interface, and specifies several pairs of an ACL to match on and the forwarding behavior.
- *Interface* – specifies which route map is relevant to it. Each physical interface, such as FastEthernet 0/48, can have a different Route Map, or if groups of physical interfaces are to have their traffic handled in the same manner (*e.g.,* if the OpenFlow rule does not specify an input port as part of the match condition), then we can optimize by assigning the interfaces all to the same VLAN, so that they are all aggregated as one interface.

To illustrate, shown below is some familiar OpenFlow syntax that is used to describe match criteria and apply a forwarding behavior. We expect to enter a flow with something like the below statement.

```
match: src_ip="1.2.3.4",
    dest_ip="2.3.4.5", action:OUT_PORT_2
```

To push a flow rule and its forwarding behavior to a switch we want to create a route-map, apply an access-control list to it, define its forwarding behavior, install an access-control entry for the access-list to match on, and finally, apply the route-map to the interfaces we require as given by our stored topology state.

So, the ClosedFlow controller creates the new route-map and defines forwarding behavior as the next-hop IP address of our adjacent switch (per the topology discovery, we know OUT_PORT_2 has a next hop address of 2.0.0.1), applies an access-list to it (ACL 101), creates the access-control entry based on the operator supplied match condition (involving the source and destination addresses), and applies the route-map to interface VLAN 1 (in this case, no match condition specifies the input port, so we default to using the default VLAN interface). Configurations applied shown below.

```
Switch1#show route-map
route-map SW1_OUTBOUND, permit, sequence 10
  Match clauses:
    ip address (access-lists): 101
  Set clauses:
    ip next-hop 2.0.0.1

Switch1#show access-lists
Extended IP access-list 101
    10 permit ip host 1.2.3.4 host 2.3.4.5

Switch1#show run interface vlan 1
interface Vlan1
 ip address 1.2.3.1 255.255.255.0
 ip policy route-map SW1_OUTBOUND
end
```

2.4 Handling Packet-In Events

The fourth capability associated with OpenFlow networks is the special action 'send to controller', which can be used, for example, to enable a reactive network where the first packet of every flow is sent to the controller.

In an OpenFlow network, when a packet arrives at an interface, we have a couple potential outcomes: the packet matches a flow table entry and some forwarding, dropping, or modify action is applied, or when the packet does not meet these criteria (*e.g.,* a "table miss"), it is sent to the controller allowing decision logic to take place. We propose two potential options to handle "table misses", or more generally 'send-to-controller' actions (which do not have to be table misses) – neither option mimics OpenFlow exactly, but has aspects of OpenFlow's two ways to handle send-to-controller.

- **Remote Logging on Explicit Deny:** Our first option uses the remote logging feature to send a message to the controller when a packet does not match our access control criteria

specified in our route-map. To achieve this, we must place an "explicit deny" access control entry (ACE) at the end of our access-control list. Additionally, we must us the "log-input" keyword to indicate that a syslog entry should be made if the explicit deny is matched on. To ensure that our remote logging is restricted to this specific message, we create a logging discriminator that uses regular expression matching and we suppress excessive logging with simple threshold limits until a flow rule is installed. An example of these configuration steps is shown below, and the down side is that while the header of the packet is sent to the controller, the packet itself is dropped (instead of being buffered at the switch, as in OpenFlow).

```
SW1(config)# access-list 101 permit
    icmp host 2.2.2.2 host 3.3.3.3
SW1(config)# access-list 101 permit
    icmp host 3.3.3.3 host 2.2.2.2
SW1(config)# access-list 101 deny udp
    any gt 0 any gt 0 log-input
SW1(config)# access-list 101 deny tcp
    any gt 0 any gt 0 log-input
SW1(config)# access-list 101 deny ip
    any any log-input
SW1(config)# logging discriminator
    tblMiss msg-body ''101 denied''
SW1(config)# logging host
    <<controller-IP>> transport udp
    port 22345 discriminator tblMiss
```

- **Send Entire Packet To Controller:** Rather than implementing remote logging to notify the controller that no match criteria were met, we also have the ability to direct the flow to the controller and allow it to decide whether the traffic will be forwarded over a particular interface or be dropped by the switches involved. To accomplish this, rather than logging an access control entry, we want to apply a "forward-to-controller.. behavior by essentially defining this as our default forwarding behavior. Once the controller receives the first packet, a decision is made to install a flow entry to either forward the traffic down a preferred path, or drop the traffic.

```
SW1(config)# access-list 103 deny ip
    any any
SW1(config)# route-map SW1_OUT 15
SW1(config-route-map)# match ip
    address 103
SW1(config-route-map)# set ip
    next-hop <<controller-IP>>
SW1(config)# int vlan 1
SW1(config-if)# ip policy route-map
    SW1_OUT
```

Neither of these options, directly matches the OpenFlow packet-in message, specifically the option where the packet is buffered on the switch and only the header is sent to the controller. In one option, we can match the behavior that only the header is sent to the controller, but the packet is dropped (relying on retransmission). In the other option, we can match the behavior of OpenFlow which sends the entire packet to the controller, which has overhead, but only for highly reactive networks (more proactive SDN control would not suffer from this overhead). We are still exploring mechanisms to realize the buffering capability to increase reliability.

3. PROTOTYPE

Our end-goal is for this to be able to be integrated into an SDN controller, and allow SDN applications to run, unmodified, controlling legacy switches instead of OpenFlow switches. Our initial prototype is not quite there yet. We instead implemented two independent programs to illustrate the key parts:

- A constantly-running topology discovery application which uses the info received from the remote logs to display the current adjacencies, and
- A simple python program equivalent to static flow pusher which allows flow modifications to be specified.

With these two capabilities, we intend to integrate a Cisco configuration backend with an SDN controller which provides an API which can closely match the OpenFlow control (*e.g.,* with the OFPTableMod method in Ryu, or with the flow entry files in *yanc* [19]). Since *yanc* completely separates out the the hardware interfacing control, it provides a simple path to allowing us to add support for the legacy devices (by creating a new driver) with no modifications to the yanc code or applications – and we can, without modifying the application, we can gradually replace legacy switches with OpenFlow enabled switches.

4. EXPERIMENT AND EVALUATION

4.1 Environment Setup

Throughout our experiments and implementation, we utilized Cisco 3550 Multi-Layer Switches with a minimum IOS of 12.2(44)SE. Our research and observations have shown that functionality and control only increase in granularity with each new evolution. In particular, we later examine Cisco's Embedded Event Manager and Tool Command Line scripting features, first introduced to the Catalyst family of switches, with the Cisco 3560 MLS using IOS version 12.2-(55)SE. These features enable vast potential for customized functionality to support communication with a central control plane.

- **Configure SDM Template:** To optimize our environment for policy-based routing and TCAM ACL entries, we must first reformat our TCAM table using the Switch Database Manager. Template options for formatting the TCAM tables include access, default (balance of all functionality), routing, and VLAN. The first of these options, access, allows us to maximize our resources for ACL functionality. We choose this template because ACL entries on layer 3 and 4 fields will act as the majority of our configuration entries when installing flow rules. To enable policy-based routing it is important that the 'extended-match' keyword is used with the SDM template enabling 144-bit layer-3 TCAM. After these commands are used, a reboot is required.

- **Enable IP Routing and Cisco Express Forwarding**: Because we wish to match on layer 3 and 4 packet fields and define interface forwarding behavior with policy-based routing, we will enable IP routing and Cisco Express Forwarding (CEF). We exploit the benefits that CEF offers through its direct use of the Forwarding Information Base and adjacency tables to perform fast IP switching coupled with Policy-based routing (PBR) Route-Maps that specify forwarding behaviors and match criteria stored in the TCAM.

4.2 Results

To evaluate the effects of our technique on the hardware and the feasibility of operation in a production network, we chose to measure the direct correlation between installed flow rules and their storage in the TCAM. We chose 3 flow rule datasets: a realistic enterprise sampling which used realistic IP address ranges, port ranges, and matching on both layer 3 and 4 fields. The other two sample datasets used a worst-case, completely random source/destination IP and source/destination port combination. With these datasets, we essentially want to see how quickly we can cause TCAM ACL installation and merge failures if an administrator does not adhere to basic, responsible configuration best practices. In each worst case event, rules are applied in software, requiring CPU processing for match criteria.

We see that our realistic network flow dataset scales quite well. The Cisco Catalyst 3550 multi-layer switches used for our experiments permitted up to 4432 PBR TCAM entries. Based on the graph in Figure 2, we may extrapolate that our flow rule limit would be 50,000 for this particular piece of hardware. In our final two randomized datasets, we see that the switch begins to experience ACL merge and installation failures at 500, for layer 3, and 250 layer 4, access-list rules respectively. These results are consistent with Cisco's recommendations to avoid TCAM resource exhaustion.

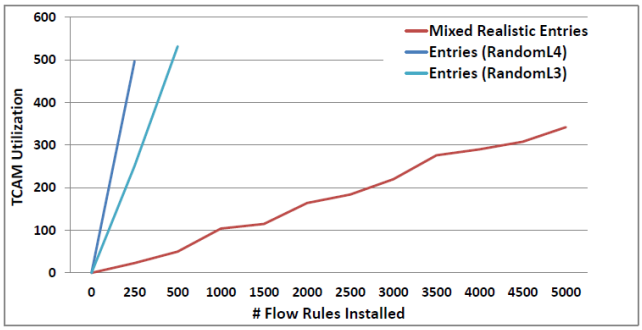

Figure 2: TCAM Policy-Based Routing Utilization

For our next evaluation, we chose to determine the impact of the previous cases where we experienced TCAM merge and installation failures, that is, the newly added flow rules would be processed in software until space became available. We performed tests using Iperf to witness forwarding rate performance at 100 percent interface utilization. Figure 3 depicts the results, by average bitrate, for flow rules installed in software, as IOS configuration that are CPU processed, and in hardware, installed as TCAM policy-based routing entries. Two flows are depicted, one installed in hardware from time zero, and another that is installed in software (because the TCAM was too full), and at 10 seconds in, can be moved to hardware as space became available (we deleted other rules). Upon transition into TCAM, we see an immediate jump to full hardware speed.

5. DISCUSSION

Looking beyond realizing OpenFlow on the specific legacy switch we used, we discuss here how we can go beyond OpenFlow and we can also go beyond the model and vendor we used.

5.1 OpenFlow Extensions

In some regards we are able to go beyond what OpenFlow provides if we allow capabilities of the legacy switches to be exposed (*i.e.,* if we don't limit ourselves to OpenFlow only). For example, AvantGuard [22], DevoFlow [8], and Software-defined counters [18] each noted some limitations with OpenFlow with regard to security or monitoring applications and proposed new switch additions. Existing legacy switches already have lots of other capabilities, such as the on-switch monitoring with triggered events along the lines of what AvantGuard and Software-defined counters proposed, namely through the use of Cisco's Embedded Event Manager [2]. With the send to controller action, we were not able to exactly match what OpenFlow can provide. But with legacy switches we can match the intent of the proposed extensions (though, likely not the exact realization proposed).

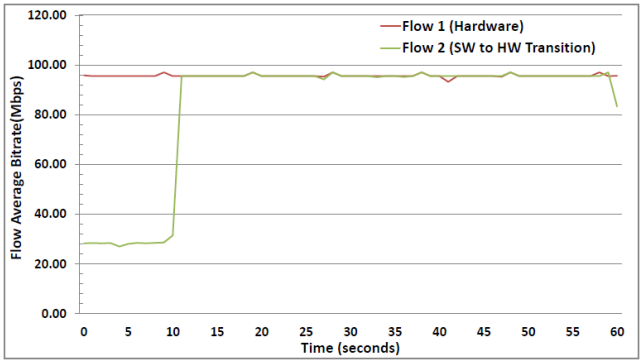

Figure 3: Flow Performance Hardware vs. Software

5.2 Other Switches (Vendors, Models)

Despite the fact that we chose to implement these techniques in Cisco 3550 and 3560 series multi-layer switches, we note the presence of identical functionality among other major vendors' operating systems – we examined HP and Juniper switches, specifically, and believe we would find it to be the case with other vendors as well.

Further, it is important to note that the functionality we describe can be realized in newer equipment models (from Cisco). Each generation beyond the equipment we used carries richer functionality and more powerful features than the previous. In more recent Cisco switches, we have the facility for logging Cisco Discovery Protocol adjacency changes or the use of the Link Layer Discovery Protocol at layer-2 for topology discovery which allows us to avoid using OSPF for control channel connectivity and communication at time-zero. In many older Cisco branch routers and slightly newer multi-layer switches, we have added packet classification granularity with the Network Based Application Recognition(NBAR) feature. This allows us to use deep packet inspection to classify traffic and make decisions up to layer-7. Cisco maintains a vast database of application signatures and NBAR allows for creating custom signatures.

6. CONCLUSION

In this paper we have discussed specific techniques for leveraging open, logically centralized, network control, and OpenFlow-like control over the forwarding plane of proprietary legacy equipment. We show that the barrier to entry by organizations may be alleviated by offering options for transitioning from a legacy network with proprietary configuration, to an all SDN network with custom control applications without the need for a costly network-wide upgrade of equipment. In addition, we discuss that technologies that have been in network equipment can be used to realize similar functionality to many of the new OpenFlow extensions proposed recently.

7. REFERENCES

[1] About nicira. http://www.nicira.com/about/.

[2] Cisco IOS Embedded Event Manager (EEM). http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-embedded-event-manager-eem/.

[3] Perspectives: Networking needs a VMware. http://www.bigswitch.com/wp/about-us/.

[4] Ryu. http://osrg.github.io/ryu/.

[5] Floodlight. http://floodlight.openflowhub.org, 2013.

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. In *Proc. SIGCOMM*, 2007.

[7] T.-C. Chiueh, C.-C. Tu, Y.-C. Wang, P.-W. Wang, K.-W. Li, and Y.-M. Huang. Peregrine: An all-layer-2 container computer network. In *IEEE Conference on Cloud Computing (CLOUD)*, June 2012.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-performance Networks. In *Proc. SIGCOMM*, 2011.

[9] D. Erickson et al. A demonstration of virtual machine mobility in an OpenFlow network, August 2008. Demo at *ACM SIGCOMM*.

[10] 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.

[11] R. Hand, M. Ton, and E. Keller. Active security. In *Proc Workshop on Hot Topics in Networks (HotNets)*, 2013.

[12] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. Plug-n-Serve: Load-balancing web traffic using OpenFlow, August 2009. Demo at *ACM SIGCOMM*.

[13] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: Saving energy in data center networks. In *Networked Systems Design and Implementation*, April 2010.

[14] L. Jose, M. Yu, and J. Rexford. Online measurement of large traffic aggregates on commodity switches. In *Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, March 2011.

[15] 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 *Operating Systems Design and Implementation*, October 2010.

[16] D. Levin, M. Canini, S. Schmid, and A. Feldmann. Panopticon: Reaping the Benefits of Partial SDN Deployment in Enterprise Networks. Technical report, Technische Universität Berlin / Deutsche Telekom Laboratories, 2013.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. (CCR)*, 38(2), 2008.

[18] J. C. Mogul and P. Congdon. Hey, You Darned Counters!: Get off My ASIC! In *Proc. Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[19] M. Monaco, O. Michel, and E. Keller. Applying Operating System Principles to SDN Controller Design. In *Proc. Workshop on Hot Topics in Networks (HotNets)*, 2013.

[20] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Operating Systems Design and Implementation*, October 2010.

[21] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *Proc. Network and Distributed System Security Symposium (NDSS)*, February 2013.

[22] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks. In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[23] R. Wang, D. Butnariu, and J. Rexford. OpenFlow-based server load balancing gone wild. In *Hot-ICE*, March 2011.