OpenFlow: Enabling Innovation in Campus Networks

March 14, 2008

Nick McKeown
Stanford University

Tom Anderson
University of Washington

Hari Balakrishnan
MIT

Guru Parulkar
Stanford University

Larry Peterson
Princeton University

Jennifer Rexford
Princeton University

Scott Shenker
University of California, Berkeley

Jonathan Turner
Washington University in St. Louis

ABSTRACT

This whitepaper proposes OpenFlow: a way for researchers to run experimental protocols in the networks they use every day. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal is to encourage networking vendors to add OpenFlow to their switch products for deployment in college campus backbones and wiring closets. We believe that OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI. Two buildings at Stanford University will soon run OpenFlow networks, using commercial Ethernet switches and routers. We will work to encourage deployment at other schools; and We encourage you to consider deploying OpenFlow in your university network too.

Categories and Subject Descriptors

C.2 [**Internetworking**]: Routers

General Terms

Experimentation, Design

Keywords

Ethernet switch, virtualization, flow-based

1. THE NEED FOR PROGRAMMABLE NETWORKS

Networks have become part of the critical infrastructure of our businesses, homes and schools. This success has been both a blessing and a curse for networking researchers; their work is more relevant, but their chance of making an impact is more remote. The reduction in real-world impact of any given network innovation is because the enormous installed base of equipment and protocols, and the reluctance to experiment with production traffic, which have created an exceedingly high barrier to entry for new ideas. Today, there is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment. The result is that most new ideas from the networking research community go untried and untested; hence the commonly held belief that the network infrastructure has "ossified".

Having recognized the problem, the networking community is hard at work developing programmable networks, such as GENI [1] a proposed nationwide research facility for experimenting with new network architectures and distributed systems. These programmable networks call for programmable switches and routers that (using *virtualization*) can process packets for multiple isolated experimental networks simultaneously. For example, in GENI it is envisaged that a researcher will be allocated a *slice* of resources across the whole network, consisting of a portion of network links, packet processing elements (e.g. routers) and end-hosts; researchers program their slices to behave as they wish. A slice could extend across the backbone, into access networks, into college campuses, industrial research labs, and include wiring closets, wireless networks, and sensor networks.

Virtualized programmable networks could lower the barrier to entry for new ideas, increasing the rate of innovation in the network infrastructure. But the plans for nationwide facilities are ambitious (and costly), and it will take years for them to be deployed.

This whitepaper focuses on a shorter-term question closer to home: *As researchers, how can we run experiments in our campus networks?* If we can figure out how, we can start soon and extend the technique to other campuses to benefit the whole community.

To meet this challenge, several questions need answering, including: In the early days, how will college network administrators get comfortable putting experimental equipment (switches, routers, access points, etc.) into their network? How will researchers control a portion of their local network in a way that does not disrupt others who depend on it? And exactly what functionality is needed in network switches to enable experiments? Our goal here is to propose a new switch feature that can help extend programmability into the wiring closet of college campuses.

One approach - that we do not take - is to persuade

commercial "name-brand" equipment vendors to provide an open, programmable, virtualized platform on their switches and routers so that researchers can deploy new protocols, while network administrators can take comfort that the equipment is well supported. This outcome is very unlikely in the short-term. Commercial switches and routers do not typically provide an open software platform, let alone provide a means to virtualize either their hardware or software. The practice of commercial networking is that the standardized external interfaces are narrow (i.e., just packet forwarding), and all of the switch's internal flexibility is hidden. The internals differ from vendor to vendor, with no standard platform for researchers to experiment with new ideas. Further, network equipment vendors are understandably nervous about opening up interfaces inside their boxes: they have spent years deploying and tuning fragile distributed protocols and algorithms, and they fear that new experiments will bring networks crashing down. And, of course, open platforms lower the barrier-to-entry for new competitors.

A few open software platforms already exist, but do not have the performance or port-density we need. The simplest example is a PC with several network interfaces and an operating system. All well-known operating systems support routing of packets between interfaces, and open-source implementations of routing protocols exist (e.g., as part of the Linux distribution, or from XORP [2]); and in most cases it is possible to modify the operating system to process packets in almost any manner (e.g., using Click [3]). The problem, of course, is performance: A PC can neither support the number of ports needed for a college wiring closet (a fanout of 100+ ports is needed per box), nor the packet-processing performance (wiring closet switches process over 100Gbits/s of data, whereas a typical PC struggles to exceed 1Gbit/s; and the gap between the two is widening).

Existing platforms with specialized hardware for line-rate processing are not quite suitable for college wiring closets either. For example, an ATCA-based virtualized programmable router called the Supercharged PlanetLab Platform [4] is under development at Washington University, and can use network processors to process packets from many interfaces simultaneously at line-rate. This approach is promising in the long-term, but for the time being is targeted at large switching centers and is too expensive for widespread deployment in college wiring closets. At the other extreme is NetFPGA [5] targeted for use in teaching and research labs. NetFPGA is a low-cost PCI card with a user-programmable FPGA for processing packets, and 4-ports of Gigabit Ethernet. NetFPGA is limited to just four network interfaces—insufficient for use in a wiring closet.

Thus, the commercial solutions are too closed and inflexible, and the research solutions either have insufficient performance or fanout, or are too expensive. It seems unlikely that the research solutions, with their complete generality, can overcome their performance or cost limitations. A more promising approach is to compromise on generality and to seek a degree of switch flexibility that is:

- Amenable to high-performance and low-cost implementations.
- Capable of supporting a broad range of research.
- Assured to isolate experimental traffic from production traffic.
- Consistent with vendors' need for closed platforms.

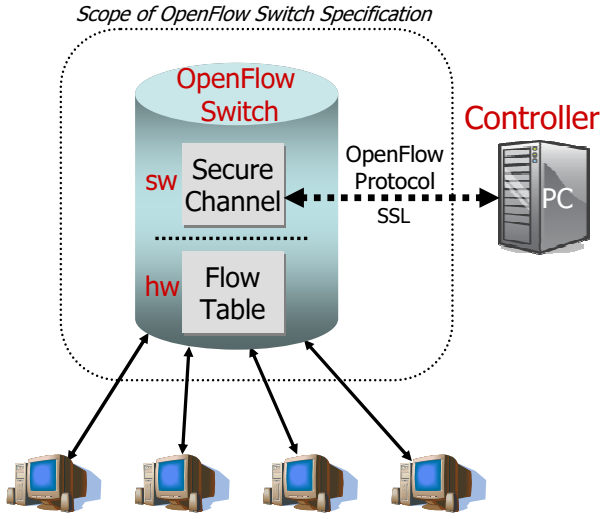

Figure 1: Idealized OpenFlow Switch. The Flow Table is controlled by a remote controller via the Secure Channel.

This paper describes the OpenFlow Switch—a specification that is an initial attempt to meet these four goals.

2. THE OPENFLOW SWITCH

The basic idea is simple: we exploit the fact that most modern Ethernet switches and routers contain flow-tables (typically built from TCAMs) that run at line-rate to implement firewalls, NAT, QoS, and to collect statistics. While each vendor's flow-table is different, we've identified an interesting common set of functions that run in many switches and routers. OpenFlow exploits this common set of functions.

OpenFlow provides an open protocol to program the flow-table in different switches and routers. A network administrator can partition traffic into production and research flows. Researchers can control their own flows - by choosing the routes their packets follow and the processing they receive. In this way, researchers can try new routing protocols, security models, addressing schemes, and even alternatives to IP. On the same network, the production traffic is isolated and processed in the same way as today.

The datapath of an OpenFlow Switch consists of a Flow Table, and an action associated with each flow entry. The set of actions supported by an OpenFlow Switch is extensible, but below we describe a minimum requirement for all switches. For high-performance and low-cost the datapath must have a carefully prescribed degree of flexibility. This means forgoing the ability to specify arbitrary handling of each packet and seeking a more limited, but still useful, range of actions. Therefore, later in the paper, define a basic required set of actions for all OpenFlow switches.

An OpenFlow Switch consists of at least three parts: (1) A *Flow Table*, with an action associated with each flow entry, to tell the switch how to process the flow, (2) A *Secure Channel* that connects the switch to a remote control process (called the *controller*), allowing commands and packets

to be sent between a controller and the switch using (3) The *OpenFlow Protocol*, which provides an open and standard way for a controller to communicate with a switch. By specifying a standard interface (the OpenFlow Protocol) through which entries in the Flow Table can be defined externally, the OpenFlow Switch avoids the need for researchers to program the switch.

It is useful to categorize switches into dedicated OpenFlow switches that do not support normal Layer 2 and Layer 3 processing, and OpenFlow-enabled general purpose commercial Ethernet switches and routers, to which the OpenFlow Protocol and interfaces have been added as a new feature.

Dedicated OpenFlow switches. A dedicated OpenFlow Switch is a dumb datapath element that forwards packets between ports, as defined by a remote control process. Figure 1 shows an example of an OpenFlow Switch.

In this context, flows are broadly defined, and are limited only by the capabilities of the particular implementation of the Flow Table. For example, a flow could be a TCP connection, or all packets from a particular MAC address or IP address, or all packets with the same VLAN tag, or all packets from the same switch port. For experiments involving non-IPv4 packets, a flow could be defined as all packets matching a specific (but non-standard) header.

Each flow-entry has a simple action associated with it; the three basic ones (that all dedicated OpenFlow switches must support) are:

1. Forward this flow's packets to a given port (or ports). This allows packets to be routed through the network. In most switches this is expected to take place at linerate.

2. Encapsulate and forward this flow's packets to a controller. Packet is delivered to Secure Channel, where it is encapsulated and sent to a controller. Typically used for the first packet in a new flow, so a controller can decide if the flow should be added to the Flow Table. Or in some experiments, it could be used to forward all packets to a controller for processing.

3. Drop this flow's packets. Can be used for security, to curb denial of service attacks, or to reduce spurious broadcast discovery traffic from end-hosts.

An entry in the Flow-Table has three fields: (1) A packet header that defines the flow, (2) The action, which defines how the packets should be processed, and (3) Statistics, which keep track of the number of packets and bytes for each flow, and the time since the last packet matched the flow (to help with the removal of inactive flows).

In the first generation "Type 0" switches, the flow header is a 10-tuple shown in Table 1. A TCP flow could be specified by all ten fields, whereas an IP flow might not include the transport ports in its definition. Each header field can be a wildcard to allow for aggregation of flows, such as flows in which only the VLAN ID is defined would apply to all traffic on a particular VLAN.

The detailed requirements of an OpenFlow Switch are defined by the *OpenFlow Switch Specification* [6].

OpenFlow-enabled switches. Some commercial switches, routers and access points will be enhanced with the OpenFlow feature by adding the Flow Table, Secure Channel and OpenFlow Protocol (we list some examples in Section 5). Typically, the Flow Table will re-use existing hardware, such as a TCAM; the Secure Channel and Protocol will be ported to run on the switch's operating system. Figure 2 shows a network of OpenFlow-enabled commercial switches and access points. In this example, all the Flow Tables are managed by the same controller; the OpenFlow Protocol allows a switch to be controlled by two or more controllers for increased performance or robustness.

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

Table 1: The header fields matched in a "Type 0" OpenFlow switch.

Our goal is to enable experiments to take place in an existing production network alongside regular traffic and applications. Therefore, to win the confidence of network administrators, OpenFlow-enabled switches must isolate experimental traffic (processed by the Flow Table) from production traffic that is to be processed by the normal Layer 2 and Layer 3 pipeline of the switch. There are two ways to achieve this separation. One is to add a *fourth action*:

4. Forward this flow's packets through the switch's normal processing pipeline.

The other is to define separate sets of VLANs for experimental and production traffic. Both approaches allow normal production traffic that isn't part of an experiment to be processed in the usual way by the switch. All OpenFlow-enabled switches are required to support one approach or the other; some will support both.

Additional features. If a switch supports the header formats and the four basic actions mentioned above (and detailed in the *OpenFlow Switch Specification*), then we call it a "Type 0" switch. We expect that many switches will support additional actions, for example to rewrite portions of the packet header (e.g., for NAT, or to obfuscate addresses on intermediate links), and to map packets to a priority class. Likewise, some Flow Tables will be able to match on arbitrary fields in the packet header, enabling experiments with new non-IP protocols. As a particular set of features emerges, we will define a "Type 1" switch.

Controllers. A controller adds and removes flow-entries from the Flow Table on behalf of experiments. For example, a static controller might be a simple application running on a PC to statically establish flows to interconnect a set of test computers for the duration of an experiment. In this case the flows resemble VLANs in current networks—providing a simple mechanism to isolate experimental traffic from the production network. Viewed this way, OpenFlow is a generalization of VLANs.

One can also imagine more sophisticated controllers that dynamically add/remove flows as an experiment progresses. In one usage model, a researcher might control the complete network of OpenFlow Switches and be free to decide how all flows are processed. A more sophisticated controller might support multiple researchers, each with different accounts and permissions, enabling them to run multiple independent experiments on different sets of flows. Flows identified

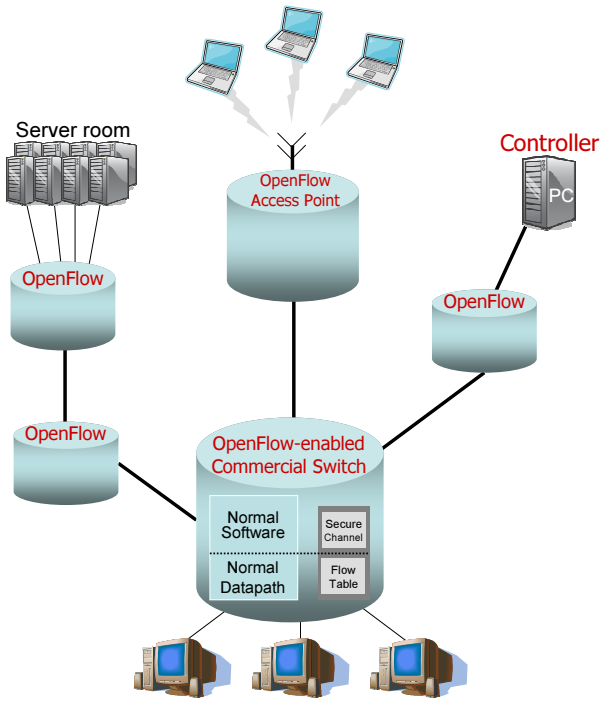

Figure 2: Example of a network of OpenFlow-enabled commercial switches and routers.

as under the control of a particular researcher (e.g., by a policy table running in a controller) could be delivered to a researcher's user-level control program which then decides if a new flow-entry should be added to the network of switches.

3. USING OPENFLOW

As a simple example of how an OpenFlow Switch might be used imagine that Amy (a researcher) invented Amy-OSPF as a new routing protocol to replace OSPF. She wants to try her protocol in a network of OpenFlow Switches, without changing any end-host software. Amy-OSPF will run in a controller; each time a new application flow starts Amy-OSPF picks a route through a series of OpenFlow Switches, and adds a flow- entry in each switch along the path. In her experiment, Amy decides to use Amy-OSPF for the traffic entering the OpenFlow network from her own desktop PC—so she doesn't disrupt the network for others. To do this, she defines one flow to be all the traffic entering the OpenFlow switch through the switch port her PC is connected to, and adds a flow-entry with the action "Encapsulate and forward all packets to a controller". When her packets reach a controller, her new protocol chooses a route and adds a new flow-entry (for the application flow) to every switch along the chosen path. When subsequent packets arrive at a switch, they are processed quickly (and at line-rate) by the Flow Table.

There are legitimate questions to ask about the performance, reliability and scalability of a controller that dynamically adds and removes flows as an experiment progresses: Can such a centralized controller be fast enough to process new flows and program the Flow Switches? What happens when a controller fails? To some extent these questions were addressed in the context of the Ethane prototype, which used simple flow switches and a central controller [7]. Preliminary results suggested that an Ethane controller based on a low-cost desktop PC could process over 10,000 new flows per second — enough for a large college campus. Of course, the rate at which new flows can be processed will depend on the complexity of the processing required by the researcher's experiment. But it gives us confidence that meaningful experiments can be run. Scalability and redundancy are possible by making a controller (and the experiments) stateless, allowing simple load-balancing over multiple separate devices.

3.1 Experiments in a Production Network

Chances are, Amy is testing her new protocol in a network used by lots of other people. We therefore want the network to have two additional properties:

1. Packets belonging to users other than Amy should be routed using a standard and tested routing protocol running in the switch or router from a "name-brand" vendor.

2. Amy should only be able to add flow entries for her traffic, or for any traffic her network administrator has allowed her to control.

Property 1 is achieved by OpenFlow-enabled switches. In Amy's experiment, the default action for all packets that don't come from Amy's PC could be to forward them through the normal processing pipeline. Amy's own packets would be forwarded directly to the outgoing port, without being processed by the normal pipeline.

Property 2 depends on the controller. The controller should be seen as a platform that enables researchers to implement various experiments, and the restrictions of Property 2 can be achieved with the appropriate use of permissions or other ways to limit the powers of individual researchers to control flow entries. The exact nature of these permission-like mechanisms will depend on how the controller is implemented. We expect that a variety of controllers will emerge. As an example of a concrete realization of a controller, some of the authors are working on a controller called NOX as a follow-on to the Ethane work [8]. A quite different controller might emerge by extending the GENI management software to OpenFlow networks.

3.2 More Examples

As with any experimental platform, the set of experiments will exceed those we can think of up-front — most experiments in OpenFlow networks are yet to be thought of. Here, for illustration, we offer some examples of how OpenFlow-enabled networks could be used to experiment with new network applications and architectures.

Example 1: Network Management and Access Control. We'll use Ethane as our first example [7] as it was the research that inspired OpenFlow. In fact, an OpenFlow Switch can be thought of as a generalization of Ethane's datapath switch. Ethane used a specific implementation of a controller, suited for network management and control, that manages the admittance and routing of flows. The basic idea of Ethane is to allow network managers to define a

network-wide policy in the central controller, which is enforced directly by making admission control decisions for each new flow. A controller checks a new flow against a set of rules, such as "Guests can communicate using HTTP, but only via a web proxy" or "VoIP phones are not allowed to communicate with laptops." A controller associates packets with their senders by managing all the bindings between names and addresses — it essentially takes over DNS, DHCP and authenticates all users when they join, keeping track of which switch port (or access point) they are connected to. One could envisage an extension to Ethane in which a policy dictates that particular flows are sent to a user's process in a controller, hence allowing researcher-specific processing to be performed in the network.

Example 2: VLANs. OpenFlow can easily provide users with their own isolated network, just as VLANs do. The simplest approach is to statically declare a set of flows which specify the ports accessible by traffic on a given VLAN ID. Traffic identified as coming from a single user (for example, originating from specific switch ports or MAC addresses) is tagged by the switches (via an action) with the appropriate VLAN ID.

A more dynamic approach might use a controller to manage authentication of users and use the knowledge of the users' locations for tagging traffic at runtime.

Example 3: Mobile wireless VOIP clients. For this example consider an experiment of a new call- handoff mechanism for WiFi-enabled phones. In the experiment VOIP clients establish a new connection over the OpenFlow-enabled network. A controller is implemented to track the location of clients, re-routing connections — by reprogramming the Flow Tables — as users move through the network, allowing seamless handoff from one access point to another.

Example 4: A non-IP network. So far, our examples have assumed an IP network, but OpenFlow doesn't require packets to be of any one format — so long as the Flow Table is able to match on the packet header. This would allow experiments using new naming, addressing and routing schemes. There are several ways an OpenFlow-enabled switch can support non-IP traffic. For example, flows could be identified using their Ethernet header (MAC src and dst addresses), a new EtherType value, or at the IP level, by a new IP Version number. More generally, we hope that future switches will allow a controller to create a generic mask (offset + value + mask), allowing packets to be processed in a researcher-specified way.

Example 5: Processing packets rather than flows. The examples above are for experiments involving flows — where a controller makes decisions when the flow starts. There are, of course, interesting experiments to be performed that require every packet to be processed. For example, an intrusion detection system that inspects every packet, an explicit congestion control mechanism, or when modifying the contents of packets, such as when converting packets from one protocol format to another.

There are two basic ways to process packets in an OpenFlow-enabled network. First, and simplest, is to force all of a flow's packets to pass through a controller. To do this, a controller doesn't add a new flow entry into the Flow Switch — it just allows the switch to default to forwarding every packet to a controller. This has the advantage of flexibility, at the cost of performance. It might provide a useful way to test the functionality of a new protocol, but is unlikely to be of much interest for deployment in a large network.

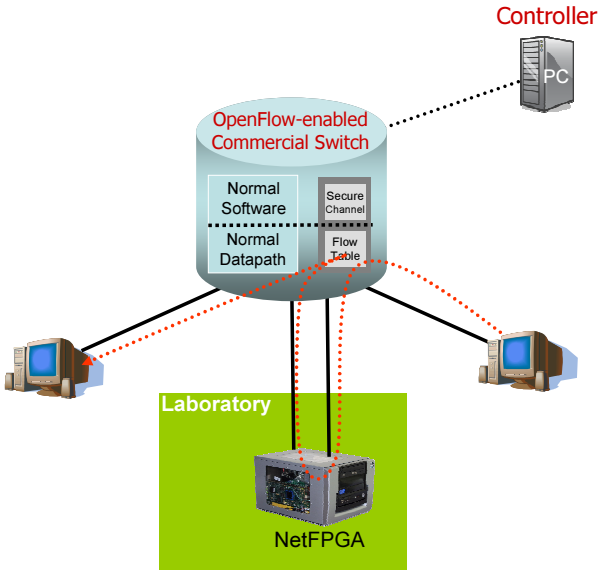

Figure 3: Example of processing packets through an external line-rate packet-processing device, such as a programmable NetFPGA router.

The second way to process packets is to route them to a programmable switch that does packet processing — for example, a NetFPGA-based programmable router. The advantage is that the packets can be processed at line-rate in a user-definable way; Figure 3 shows an example of how this could be done, in which the OpenFlow-enabled switch operates essentially as a patch-panel to allow the packets to reach the NetFPGA. In some cases, the NetFPGA board (a PCI board that plugs into a Linux PC) might be placed in the wiring closet alongside the OpenFlow-enabled switch, or (more likely) in a laboratory.

4. THE OPENFLOW CONSORTIUM

The OpenFlow Consortium aims to popularize OpenFlow and maintain the *OpenFlow Switch Specification*. The Consortium is a group of researchers and network administrators at universities and colleges who believe their research mission will be enhanced if OpenFlow-enabled switches are installed in their network.

Membership is open and free for anyone at a school, college, university, or government agency worldwide. The OpenFlow Consortium welcomes individual members who are not employed by companies that manufacture or sell Ethernet switches, routers or wireless access points (because we want to keep the consortium free of vendor influence). To join, send email to *join@OpenFlowSwitch.org*.

The Consortium web-site [1] contains the OpenFlow Switch Specification, a list of consortium members, and reference implementations of OpenFlow switches.

[1] http://www.OpenFlowSwitch.org

Licensing Model: The OpenFlow Switch Specification is free for all commercial and non-commercial use. (The exact wording is on the web-site.) Commercial switches and routers claiming to be "OpenFlow-enabled" must conform to the requirements of an OpenFlow Type 0 Switch, as defined in the OpenFlow Switch Specification. OpenFlow is a trademark of Stanford University, and will be protected on behalf of the Consortium.

5. DEPLOYING OPENFLOW SWITCHES

We believe there is an interesting market opportunity for network equipment vendors to sell OpenFlow-enabled switches to the research community. Every building in thousands of colleges and universities contains wiring closets with Ethernet switches and routers, and with wireless access points spread across campus.

We are actively working with several switch and router manufacturers who are adding the OpenFlow feature to their products by implementing a Flow Table in existing hardware; i.e. no hardware change is needed. The switches run the Secure Channel software on their existing processor.

We have found network equipment vendors to be very open to the idea of adding the OpenFlow feature. Most vendors would like to support the research community without having to expose the internal workings of their products.

We are deploying large OpenFlow networks in the Computer Science and Electrical Engineering departments at Stanford University. The networks in two buildings will be replaced by switches running OpenFlow. Eventually, all traffic will run over the OpenFlow network, with production traffic and experimental traffic being isolated on different VLANs under the control of network administrators. Researchers will control their own traffic, and be able to add/remove flow-entries.

We also expect many different OpenFlow Switches to be developed by the research community. The OpenFlow website contains "Type 0" reference designs for several different platforms: Linux (software), OpenWRT (software, for access points), and NetFPGA (hardware, 4-ports of 1GE). As more reference designs are created by the community we will post them. We encourage developers to test their switches against the reference designs.

All reference implementations of OpenFlow switches posted on the web site will be open-source and free for commercial and non-commercial use.[2]

[2]Some platforms may limit the license terms of software running on them. For example, a reference implementation on Linux may be limited by the Linux GPL.

6. CONCLUSION

We believe that OpenFlow is a pragmatic compromise that allows researchers to run experiments on heterogeneous switches and routers in a uniform way, without the need for vendors to expose the internal workings of their products, or researchers to write vendor-specific control software.

If we are successful in deploying OpenFlow networks in our campusses, we hope that OpenFlow will gradually catch-on in other universities, increasing the number of networks that support experiments. We hope that a new generation of control software emerges, allowing researchers to re-use controllers and experiments, and build on the work of others. And over time, we hope that the islands of OpenFlow networks at different universities will be interconnected by tunnels and overlay networks, and perhaps by new OpenFlow networks running in the backbone networks that connect universities to each other.

7. REFERENCES

[1] Global Environment for Network Innovations. Web site http://geni.net.

[2] Mark Handley Orion Hodson Eddie Kohler. "XORP: An Open Platform for Network Research," *ACM SIGCOMM Hot Topics in Networking, 2002.*

[3] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. "The Click modular router," *ACM Transactions on Computer Systems 18(3), August 2000, pages 263-297.*

[4] J. Turner, P. Crowley, J. Dehart, A. Freestone, B. Heller, F. Kuhms, S. Kumar, J. Lockwood, J. Lu, M.Wilson, C. Wiseman, D. Zar. "Supercharging PlanetLab - High Performance, Multi-Application, Overlay Network Platform," *ACM SIGCOMM '07, August 2007, Kyoto, Japan.*

[5] NetFPGA: Programmable Networking Hardware. Web site http://netfpga.org.

[6] The OpenFlow Switch Specification. Available at http://OpenFlowSwitch.org.

[7] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, Scott Shenker. "Ethane: Taking Control of the Enterprise," *ACM SIGCOMM '07, August 2007, Kyoto, Japan.*

[8] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casadao, Nick McKeown, Scott Shenker, "NOX: Towards an Operating System for Networks," *In submission. Also: http://nicira.com/docs/nox-nodis.pdf.*