Description of network research enablers on the example of OpenFlow

Julius Werner
(jwerner@cs.tu-berlin.de)

Seminar 'Internet Routing' ,
Technische Universität Berlin

SS 2009 (Version of June 29, 2009)

Abstract

This seminar paper describes OpenFlow, a specification developed by a research group at Stanford University that is proposed to be implemented by commercial switches and routers and would allow remote control of their forwarding behavior. It is aimed at providing researchers with an inexpensive and flexible platform to experiment with new network protocols on production-scale traffic. OpenFlow is further compared to two other projects that aim to enable network research, but differ totally in approach: the PlanetLab and the eXtensible Open Router Project (XORP). Finally, the NOX network operating system is described as an example for a project using OpenFlow's successful network hardware abstraction concept to implement a larger network management system.

1 Introduction

The Internet is an ever growing success and it does not look like that is going to slow down anytime soon. More than 500 million hosts [1] and 30 000 autonomous systems [2] are connected today and more are sure to come. But at the same time, the Internet is growing old: More than 30 years have passed since advancements in early TCP development led to the specification and implementation of the one network layer protocol that would conquer the world [3]. IP started out as part of an experimental research project to provide interconnection between several computer networks of the time, which were more heterogenous and had far fewer hosts than todays systems. The precurring TCP implementations had gone through several major versions and quite a few important changes [4]. Similarly, some ideas for the new protocol were raised and some dropped again before the final draft – for example, the often criticized 32-bit address size might have been little more than coincidence (compare proposal for variable address length in [5]). However, once the specification was implemented, it turned out to be good enough to warrant no serious further adjustments.

In the following years the Internet rapidly grew and eventually integrated commercial ISP networks. The Internet Protocol Suite was implemented many times on different systems, which dramatically increased the amount of coordination and work that would be necessary to implement further protocol changes. With the transition from the central NSFNET Backbone to a privatized, distributed backbone architecture in 1995, the Internet had finally become completely decentralized, further inhibiting architectural changes through the lack of a central coordinating instance that could propose and enforce them. The effects of this have long been evident: The most recent specification for the next generation network layer protocol has been lying in the drawer since 1998 [6] and while it has been slowly integrated into major host operating systems during the last decade, its adoption in the Internet backbone and regional ISPs is still far from global, with only a few isolated networks available. It may well be, that IPv6 is once again already outdated before it is actually adopted, for some suggest it still includes too many of IPv4's mistakes.

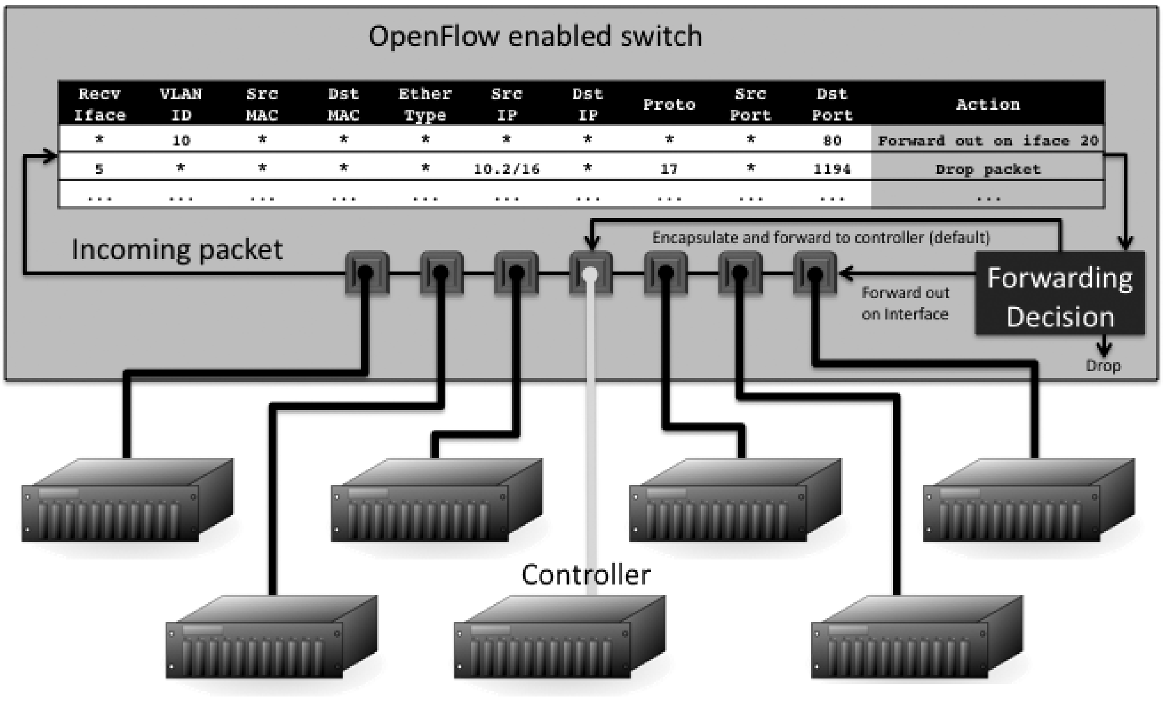
In addition to the network layer itself, these difficulties also apply to exterior gateway routing: The Border Gateway Protocol has emerged as the de facto standard for interdomain routing, and any changes to that which are not backwards-compatible would have to be conducted in the whole Internet at once. The situation here seems even worse: Todays routing tables are bursting with entries, having to cope with hundreds of thousands of prefixes [2]. While hardware vendors keep trying to increase their routers processing power to manage them, this is actually a design issue that should be solved on the whiteboard. But despite the widespread criticism of the protocol, truly innovative changes have low chances to be adopted by the Internet community [7]. Not even a standardized succession candidate (as with the Internet Protocol) has been decided upon yet. The National Research Council [8] describes this process as ossification – the reluctance to replace widespread technology with something better, because the industry is not motivated to implement, let alone develop, changes with high cost and little immediate gain, which is aggravated by the fact that pioneering the change provides no benefit until most of the competitors have joined in. Unless a good incentive to conduct individual improvements can be found, the Internet might actually need to burst apart before the long overdue corrections can be implemented.

However, before anything can be changed, there has to be a good proposal for it. It has always been the research communities' part to design these innovations, and while the IPv6 adoption may be slow, it at least shows that steady spreading of a new protocol can be possible, as long as it is generally accepted and agreed upon. But reaching widespread acceptance is a long road that involves not only innovative ideas and many refinement iterations, but also significant testing of all aspects of the new architecture, to convince the involved parties that it will hold its promises and will be worth their money. Conducting those tests is difficult though: They must be run at sufficient scale and with realistic traffic, in terms of content as well as distribution and size. With network layer and routing innovations, this requires powerful routers that can cope with backbone level traffic, a requirement usually only fulfilled by hardware-accelerated commercial network devices. But implementing experimental new protocols in those is generally not possible, since they are closed systems with proprietary firmware that allows only the range of network operations required by usual administrators. Their operating systems cannot be extended or replaced because the vendors keep their secrets under close guard and do not disclose their architectures to the public. Although there have been efforts to create dedicated research hardware that allows total control of the packet processing, that tends to be too expensive for small research groups to acquire a sufficiently large network segment. This greatly inhibits testing and demonstration of experimental network and routing protocols.

Among the most promising efforts to mitigate this problem is the OpenFlow Project. It aims at enabling large scale network protocol and routing research on the regular commercial network hardware that is usually deployed at universities and research centers. This is done by merely defining a standard for controlling packet switching decisions – it is then the hardware vendor's part to implement this into their devices. The following will describe how OpenFlow-enabled switches make their forwarding decisions and how they can be configured to offer (almost) any desired functionality.

2 Background

The basic idea of OpenFlow [9] is to create a system that grants researchers the greatest possible amount of control over the packet flow in their routers and switches, while still being easy and cheap to integrate into commercial networking hardware. In order to achieve this, packet handling



Recv Iface	VLAN ID	Src MAC	Dst MAC	Ether Type	Src IP	Dst IP	Proto	Src Port	Dst Port	Action
*	10	*	*	*	*	*	*	*	80	Forward out on iface 20
5	*	*	*	*	10.2/16	*	17	*	1194	Drop packet
...	...	...	...	...	...	...	...	...	...	...

Figure 1: OpenFlow enabled switch with Flow Table. One of the attached devices acts as the controller.

decisions are based on a common subset of the information that different switches extract from a packet during its processing: the interface it was received on and the most basic contents of its Ethernet, IP and TCP/UDP header (when they are appliable). OpenFlow-enabled devices store tuples of this data in their ‘Flow Table’ and associate them with an action, e.g. dropping the packet or sending it out on a specific interface. For further flexibility, the Flow Table keys may contain wildcards, so that “send all packets from any interface with VLAN ID 10 (taken from Ethernet Header) and destination port 80 (taken from TCP/UDP Header) out on interface 20” would translate to a valid Flow Table entry. An illustration of an example network including a switch with such an entry can be seen in Figure 1.

One of OpenFlow’s goals is that researchers can conduct experiments right on the existing production hardware of their campus networks. This requires a strict separation of experimental and production traffic in the processing rules of a switch. To minimize the impact on existing production networks, the OpenFlow specification optionally allows the “forward this packet through the switch’s normal processing pipeline” action in Flow Table entries. This allows network administrators to run their OpenFlow-enabled switch with a default rule using this action for every packet and then define exceptions for special experiments, which might be recognized by a certain VLAN ID or EtherType value and handled differently. One should note that even if this action is not supported, the Flow Table concept itself could easily be used to simulate the default behavior of a switch or router.

In order to manage the Flow Table, OpenFlow defines a secure (SSL-based) network protocol that connects the OpenFlow-enabled switch or router with a controller. The controller is a server that can remotely add or remove entries from the switch’s Flow Table. However, in order to dynamically manage flows the communication has to go both ways: the controller has to be informed about unhandled packets for which a routing decision has yet to be made. This is achieved by another Flow Table action: “encapsulate this packet and send it to the controller”. This facility makes it possible to test even complex new routing protocols on any OpenFlow-enabled switch, because all the control packets can be forwarded to and processed at the controller, which generates Flow Table entries from them and programs the switch accordingly. The net effect is a combination of the programmability and conceptual limitlessness of general purpose computers with the line-rate pure processing power of commercial, enterprise-level networking hardware.

To complete the circle, the controller can encapsulate arbitrary packets and have the switch send them out on a chosen interface. One controller can operate multiple switches for a true centralization of the network logic (while still keeping the processing workload in the switches!), but a switch may also receive commands from multiple controllers for load balancing and redundancy. As the implementation and functionality of the controller is not constrained beyond compatibility with the protocol's instructions, controllers can have arbitrary complexity. Over time, powerful frameworks that multiplex the experiments of many researchers with different access rights onto the same network might be developed and even the concept of a comprehensive, distributed 'operating system of the network' becomes possible (see Section 4).

The OpenFlow Specification is developed and maintained by the OpenFlow Consortium – an open group of researchers and administrators of campus networks. The specification consists only of the OpenFlow network protocol (and the resulting Flow Table layout) itself, because the implementation on the network hardware will be closed and vendor specific. The Consortium therefore has to rely on vendors willing to adopt their concept and add OpenFlow-compatibility to their operating systems. However, due to the small and commonly used set of elements a Flow Table key consists of, the developers are confident that most of the existing target hardware could be OpenFlow-enabled with just a firmware upgrade and several vendors already added OpenFlow to their devices on an experimental basis [10]. Right now, the OpenFlow Specification is still in a late draft phase and does not yet advise widespread implementation in network hardware, because it may still be subject to change [11].

3 Comparison to other approaches

The need for realistic network testing environments which produce significant results is not new and many approaches were and are currently being developed to fulfill that demand. These approaches often differ greatly in assumptions and target use cases, which results in many dissimilar ideas, although most of them are useful in a certain situation. It would be wise for a network researcher to consider each of the testing methods and environments, only two of which are outlined here, thoroughly for their suitability to his/her project.

Within this diversity, OpenFlow represents a compromise that tries to provide realistic test environments while staying cheap enough to be realizable. It creates a testbed in an existing single, confined network or even just part of a network while being able to simultaneously keep up the usual network service without impairment. Its most important advantage is that once OpenFlow-enabled firmware upgrades are readily available, network administrators can easily turn their existing campus networks into OpenFlow testbeds. This requires low costs from the hardware vendors and none from the users. But to achieve this ease of deployment OpenFlow has to make some trade-offs: While packets can be switched at line rate, the forwarding decision is limited by the information available through the Flow Table. While packets can be arbitrarily processed by forwarding them to the controller, the resulting overhead is so large that this must be limited to a small number of packets (like a new routing protocol or a small stream of experimental data within a production network). And the conceptual limit to a single, local network makes OpenFlow unfeasible for realistic test deployments of global-scale protocols. It is certainly possible to send traffic between multiple independent OpenFlow networks over todays Internet, but in order to test routing protocols, every hop on the path has to play its part – and the chances of convincing several Tier 1 ISPs to grant researchers access to their routers, even through the traffic separation of OpenFlow, seem slim.

3.1 PlanetLab

3.1.1 Background

PlanetLab [12] is a single, global-scale testbed which is maintained by the PlanetLab Consortium, a group of academic, industrial and government institutions that each have to run at least two nodes in PlanetLab as part of their membership. Each of these nodes offer an amount of resources like network bandwidth, processing power and memory, which can be used to instantiate virtual machines running a pre-installed but extensible Linux 2.4 kernel. What makes PlanetLab special is the concept of 'distributed virtualization': a user reserves resources on multiple nodes at once, called a slice, which are interconnected through a network overlay. This way, a whole distributed network can be reserved for an experiment, even though many other such experiments may run on the same nodes simultaneously through virtualization. Another interesting aspect of PlanetLab is that the management system itself is just another service running on a slice. With this approach, the PlanetLab core system provides only very basic management functions (like user authentication) and all higher level organization can be done transparently, interchangeably and with several independent parallel services – a concept the designers call 'unbundled management'.

Since its announcement in 2002, PlanetLab has grown significantly in size and accumulated more than 1000 nodes at nearly 500 sites [13]. In addition many experimental services have been deployed over it, including routing overlays, content distribution networks, network embedded storage and QoS overlays to name only a few [14]. Many of these have been so successful that they basically developed into production services – thus PlanetLab has become not only a research but also a deployment platform, which was partially anticipated by its designers [15].

3.1.2 Comparison to OpenFlow

PlanetLab's greatest strength lies in its size: It is a viable platform to run global routing experiments or test services meant for planet-wide interaction. Because of the nodes global distribution, a PlanetLab node is always just a few hops away in most parts of the world – if the nodes are programmed to act as gateways, one can actually offer the whole world to take part in the experiment. The functionality is unlimited due to the researchers having full control of the processes running on each node in their slice. Its greatest weakness, however, is also its size: building and maintaining all the nodes is expensive. The Consortium's institutions need to pay continuously for the research possibilities offered, which is probably the reason they only grant access to their own researchers. Another weakness arises from the virtualization and overlay concepts: the significance of performance and QoS testing is limited, because the available processing power has to be shared with other slices and the packets are tunneled through the normal Internet with all its unpredictable behavior and traffic spikes.

Compared to OpenFlow, PlanetLab is targeted at different types of network research. OpenFlow is designed to research network or transport layer innovations in a realistic setting with commonly used hardware and production size traffic. PlanetLab's intent is on testing application or service layer innovations on a global scale, with more focus on functionality and scaling than performance. The two meet at the evaluation of new routing protocols, where OpenFlow is better suited for Interior Gateway Protocols targeted at single networks and PlanetLab for global-scale Exterior Gateway Protocols.

3.2 XORP

3.2.1 Background

Another recent project intending to ease network research is the eXtensible Open Router Platform – in short: XORP [16]. XORP is an open source software router implementation aimed for maximum support of existing routing protocols as well as total extensibility of every aspect of the router. This means that not only modules for new routing protocols, but also changes to the inner workings of the forwarding queue can easily be programmed and loaded into XORP. In addition, it is designed to be as robust as possible, including the separation of multiple loaded modules from the core package, so that bugs in one extension are less likely to crash the whole router. The designers included support for all functionality commonly expected today from an Internet core router. XORP can be compiled for most UNIX based operating systems and Microsoft Windows Server, but less supported systems might not be able to use all features. Although these operating systems usually bring their own routing software, it outperforms them by far in functionality as well as performance.

The XORP code itself focuses on providing an abstract, high-level API to its extensions and a rich library of state-of-the-art routing functionality and protocol support – the low level packet processing architecture is taken from the Click Modular Router [17]. Click is a modular packet forwarding framework that builds its forwarding information base in form of a graph connecting simple processing elements. Each element provides a single, simple processing function and the flow of a packet through several elements along the graph provides the compilation of those simple functions into a complex routing process. Click provides a remarkable processing performance: in a benchmark of the maximum loss-free forwarding rate on a 700 MHz Intel Pentium III, Click (v1.1) managed 333 000 packets per second, compared to a standard Linux kernel (v2.2.14) with 84 000 packets per second. This is mostly due to Click's DMA polling architecture, totally eliminating Linux' time-expensive network hardware interrupts.

3.2.2 Comparison to OpenFlow

XORP offers a great tool for early conceptual and functional testing of new routing protocols and algorithms. The PC platform and the extensibility approach offer limitless functionality and easy setup of development testbeds. However, a software-based router cannot possibly reach the performance necessary for line-rate processing of the production traffic in a large network. Although XORP performs much better than the usual network stacks included in operating systems, it utterly succumbs to the pure forwarding power of hardware-accelerated commercial routers with their TCAMs returning forwarding entries in nanoseconds. Due to this limitation, XORP based routers cannot be used for large scale performance testing of routing or QoS protocols. The modular design might allow some of the forwarding path architecture to be moved to hardware with little effort, but packet forwarding hardware is generally only found in commercial routers and switches, which do not allow or support the installation of custom operating systems and software. Therefore, until the availability of affordable routing hardware with an open architecture, hardware-accelerated XORPs stay a theoretical exercise.

This unfeasibility of an open, extensible routing system with line-rate performance is exactly what motivated OpenFlow. With OpenFlow researchers can use a simple off-the-shelf switch to run their experiments with more traffic than a XORP system could ever hope to manage. The disadvantage is that the possible functionality of OpenFlow is limited by the columns in the Flow Table and the small bandwidth of the controller channel. It would therefore be impossible to perform production scale testing of a router for a totally new layer 3 protocol in an OpenFlow network, because every packet has to be switched according to its layer 3 address. Usually this is the IPv4 address and thus a Flow Table key, but for other protocols the switch cannot keep different packet destinations apart and forwarding every packet to the controller to make that decision is impossible due to bandwidth and controller processing power limitations. In contrast, a XORP router (with appropriate extensions) would be able to route that traffic at about the same rate as normal IPv4.

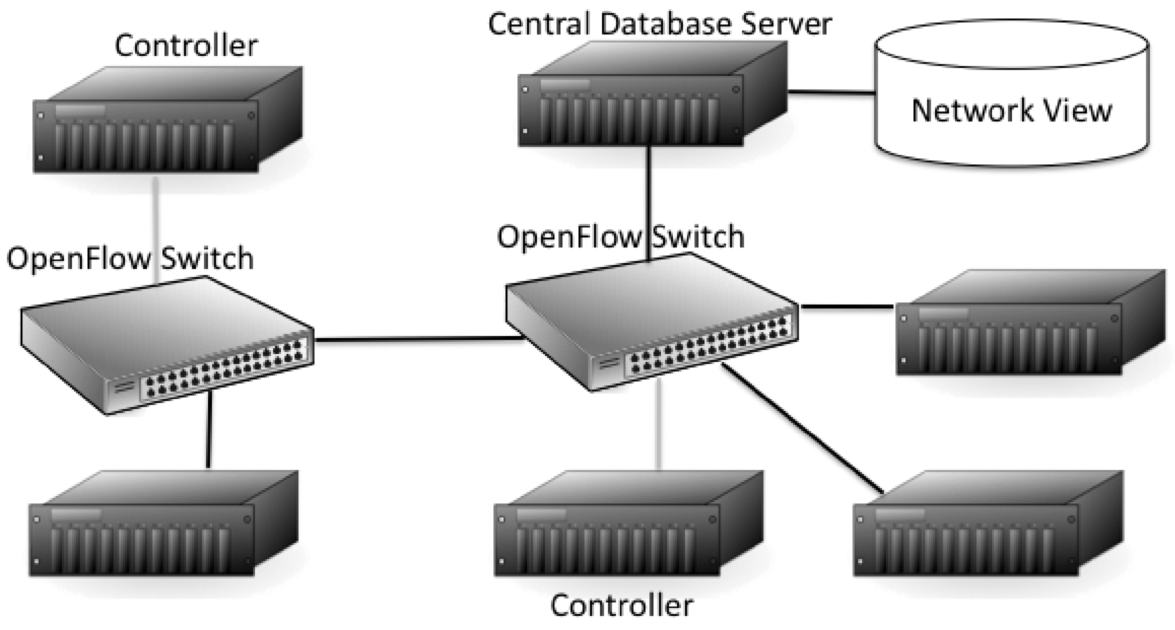


Figure 2: Example NOX setup. Note how every switch has its own controller, but state is stored centrally.

4 Further use: NOX

OpenFlow had been envisioned and designed to enable network research and testing and has grown to become a very flexible but still affordable network hardware abstraction and management framework. Its success has led to developers with non-experimental uses in mind discover its functionality. Having control of a switch's forwarding decisions from a remote controller in a standardized pattern and decoupled from the hardware implementation offers totally new possibilities in the area of centralized, abstract network management. One project that has been built upon OpenFlow is the NOX network operating system [18]. The term 'network operating system' is used to describe that NOX offers for the network what usual operating systems offer for the computer. Operating systems provide abstraction layers for underlying resources like hardware peripherals and memory, thus offering applications an execution environment, while multiplexing several applications to those underlying resources and managing their interactions. In a similar manner, NOX intends to provide abstraction layers for the underlying network infrastructure (i.e. the switches and routers), offering 'network applications' an execution environment, while implementing the same multiplexing and management concepts between them. An application is then able to work with high-level abstractions of network elements, like users and hostnames, while NOX transparently maps them to the underlying identifiers (in this example, IP addresses).

NOX runs as a single central logical instance in a network, although it may be distributed to multiple physical instances for scalability. These instances are servers running the NOX software, which acts as an OpenFlow controller to nearby routers and switches. In order to achieve consistent behavior across the network, NOX keeps its state information in a centralized database called the 'network view'. There the network topology is stored, including the abstractions like users and hostnames as well as the underlying low-level information like MAC and IP-addresses. In addition, it might also be used to keep global application state information, like policies to be enforced in the network. To create and update this network view, every NOX controlled OpenFlow switch forwards all packets from certain control protocols, like DNS, DHCP and RADIUS, to its controller, who in turn processes them to filter out information indicating a topology change, with which it updates the central database. An illustration of a possible (small) NOX network setup can be seen in Figure 2.

To support maximum scalability, NOX separates interaction with network traffic into a three layer model. The highest layer represents changes in the network view. These are determined by the controllers as described above and then, by means of the central database, propagated to the whole system. Topology changes are thus an expensive (in terms of overhead) operation, but stay manageable due to their rarity. On the middle layer are what the NOX developers termed 'flow initiations'. A flow generally represents a single, end-to-end application specific transmission, like a TCP connection, and can be identified by a Flow Table entry. The first packet of a flow, called flow initiation, is always forwarded to a controller. There its source and destination are determined and it may trigger NOX applications, which might decide that the flow should be denied, forwarded along its intended path or maybe rerouted transparently to a proxy server. When a path for it has been chosen, the controller adds appropriate entries in the affected switches' Flow Tables, so that any further packets for this specific flow can be processed by the switches themselves and don't need to bother the controller again. All necessary flow setup computations happen on a single controller, with no need to inform the system as a whole, which greatly contributes to scalability: should there be more flow initiations per second than the system can handle, one would simply add another controller server to the vicinity of the congested switch. The lowest layer of the NOX model represents all packets in the network. These are generally already part of a configured flow, so their processing happens solely inside the switches without interaction from a controller. This way, the bulk of the traffic is handled by fast, hardware-accelerated devices, with only the small fraction of them which constitute flow initiations ever being processed by the software-based controllers. This categorization of traffic is what enables NOX to utilize the processing performance of commercial network hardware, even though it is a distributed software system with centralized information and nearly unlimited, extensible functionality.

NOX itself only provides its architecture, abstraction layers and some useful basic library functions. Really utilizing its power requires running applications on top of it. Applications are simply C++ or Python programs that install event handlers in NOX. They run individually on every controller, using the NOX API to draw information from the network view (in addition to what is available from the event) and once again use the API to initiate Flow Table changes in all affected switches. A simple example application demonstrated in [18] would hook into NOX' user authentication and whenever a new user authenticates to the system confine his flows to a specific VLAN. More advanced applications might communicate with each other and even between controllers to provide complex functionality. It is quite conceivable that with an appropriate system of applications, all of todays network management and configuration task could be moved to NOX and there automated, abstracted and condensed into nice, colorful GUIs. Imagine just plugging your hosts, switches and routers together and having NOX automatically configure the desired network architecture. Administrators could simply define the abstract policies for the network and lean back while NOX configures all devices to work together harmonically – and if a new switch needs to be added, it is plugged in and NOX extends the system's architecture and policies onto it automatically. The project will still have to go a long way to reach this spectrum of functionality, but its potential already looks very promising.

5 Conclusion

The OpenFlow specification is a very helpful innovation enabling researchers and network administrators to use the capabilities of their commercial network hardware to its fullest, not only in research application but also in general network management. As a standardized interface to define packet processing rules remotely and independently from the hardware vendor, OpenFlow enables centralized network management at a scale that was formerly impossible, or only realizable through proprietary protocols and limitation on a single vendor. The critical question is whether hardware vendors would be willing to implement OpenFlow in the first place – but due to its current success, the 'OpenFlow-enabled' logo may quickly become a valuable selling proposition, once the specification will actually be finished and released for general implementation.

Several other approaches to enable network research were and are currently being developed, but they complement rather than rival each other, either focusing on different phases of testing or on different fields of research. They might even end up being combined, as the OpenFlow developers expressed in their paper the hope that Stanford's OpenFlow network (and maybe others) would be integrated into the upcoming global scale testbed GENI [19], which was inspired by PlanetLab.

Through the use of OpenFlow to shape the flows of a network, the already waning difference between routers and switches further decreases, since from OpenFlow's point of view they are the same, both offering a Flow Table to control their packet switching behavior. Proceeding further to whole networks controlled by a single, centralized network operating system, like NOX, the difference finally vanishes completely, because all former (interior) routing behavior is superseded by the Flow Tables.

References

[1] DOUGLAS E. COMER. *Computer Networks and Internets.* Prentice Hall, April 2008.

[2] TONY BATES, PHILIP SMITH, AND GEOFF HUSTON. CIDR report. http://www.cidr-report.org/as2.0/, May 2009.

[3] JON POSTEL. IEN 44: Latest header formats. ftp://ftp.isi.edu/in-notes/ien/scanned/ien44.pdf, June 1978.

[4] DAVID D. CLARK. The design philosophy of the DARPA internet protocols. 1988.

[5] JON POSTEL. IEN 66: TCP meeting notes 13. & 14. october 1977. ftp://ftp.isi.edu/in-notes/ien/scanned/ien66.pdf, October 1977.

[6] S. DEERING AND R. HINDEN. RFC 2460: Internet protocol, version 6 (IPv6) specification. December 1998.

[7] GLENN CARL AND GEORGE KESIDIS. Large-scale testing of the internet's border gateway protocol (BGP) via topological scale-down. *ACM Trans. Model. Comput. Simul.*, 18(3):1–30, 2008.

[8] NATIONAL RESEARCH COUNCIL. *Looking over the fence at networks.* National Academies Press, Washington D.C., June 2001.

[9] N. MCKEOWN, T. ANDERSON, H. BALAKRISHNAN, G. PARULKAR, L. PETERSON, J. REXFORD, S. SHENKER, AND J. TURNER. OpenFlow: enabling innovation in campus networks. 2008.

[10] JAD NAOUS, DAVID ERICKSON, G. ADAM COVINGTON, GUIDO APPENZELLER, AND NICK MCKEOWN. Implementing an OpenFlow switch on the NetFPGA platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9, San Jose, California, 2008. ACM.

[11] BRANDON HELLER. OpenFlow switch specification version 0.8.9, December 2008.

[12] LARRY PETERSON, TOM ANDERSON, DAVID CULLER, AND TIMOTHY ROSCOE. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[13] PlanetLab | an open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org/, May 2009.

[14] ANDY BAVIER, MIC BOWMAN, BRENT CHUN, DAVID CULLER, SCOTT KARLIN, STEVE MUIR, LARRY PETERSON, TIMOTHY ROSCOE, TAMMO SPALINK, AND MIKE WAWRZONIAK. Operating system support for planetary-scale network services. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 19–19, San Francisco, California, 2004. USENIX Association.

[15] LARRY PETERSON AND VIVEK S. PAI. Experience-driven experimental systems research. *Commun. ACM*, 50(11):38–44, 2007.

[16] MARK HANDLEY, ORION HODSON, AND EDDIE KOHLER. XORP: an open platform for network research. *SIGCOMM Comput. Commun. Rev.*, 33(1):53–57, 2003.

[17] ROBERT MORRIS, EDDIE KOHLER, JOHN JANNOTTI, AND M. FRANS KAASHOEK. The click modular router. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 217–231, Charleston, South Carolina, United States, 1999. ACM.

[18] NATASHA GUDE, TEEMU KOPONEN, JUSTIN PETTIT, BEN PFAFF, MARTÃN CASADO, NICK MCKEOWN, AND SCOTT SHENKER. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, 2008.

[19] GENI. http://www.geni.net/, May 2009.