[Proceeding] Openflow switching: data plane performance

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2010 proceedings

OpenFlow Switching: Data Plane Performance

Andrea Bianco, Robert Birke, Luca Giraudo, Manuel Palacin
Dipartimento di Elettronica, Politecnico di Torino, Italy
Email: {andrea.bianco, robert.birke, luca.giraudo}@polito.it, manuelpalacin@gmail.com

***Abstract*—OpenFlow is an open standard that can be implemented in Ethernet switches, routers and wireless access points (AP). In the OpenFlow framework, packet forwarding (data plane) and routing decisions (control plane) run on different devices. OpenFlow switches are in charge of packet forwarding, whereas a controller sets up switch forwarding tables on a per-flow basis, to enable flow isolation and resource slicing. We focus on the data path and analyze the OpenFlow implementation in Linux based PCs. We compare OpenFlow switching, layer-2 Ethernet switching and layer-3 IP routing performance. Forwarding throughput and packet latency in underloaded and overloaded conditions are analyzed, with different traffic patterns. System scalability is analyzed using different forwarding table size, and fairness in resource distribution is measured.**

I. Introduction

Networks are a critical element for companies and institutions, because a network failure may dramatically influence business activities. Therefore, network administrators are struggling to ensure high network availability. On the other hand, there is an increasing need for researchers to experiment on real networks new protocols and technologies. However, running experiments on the production network may be risky, and deploying a separate research network is often too costly and does not allow researchers to scale the network to large size. Programmable networks and network virtualization are just two among the technologies that may help in solving this dilemma, allowing researchers to operate on real networks without challenging production traffic or network availability.

The OpenFlow technology allows network administrators to segment telecommunication networks by properly programming network devices. OpenFlow devices identify traffic flows following forwarding rules established by network managers. OpenFlow provides a flow-based network virtualization framework to avoid interferences among different flows. Furthermore, the network administrator can delegate the management of network segment/s to the researchers, as if it was a completely independent network. Finally, OpenFlow switches are deployed into UNIX/Linux platforms and available in Ethernet switches/routers from different vendors.

Virtualization is a key element to provide flow isolation. Other technologies, like VPN (Virtual Private Network). Ethernet VLANs (IEEE 802.1Q), MPLS tunneling, or even Frame Relay and ATM, provide flow isolation. However, they often lack easy management or flexibility, being tailored to a specific set of protocols. Some research projects, e.g. GENI [1], FEDERICA [2], try to develop a programmable network in which a researcher can obtain a slice of resources (network, CPU, RAM, disk...) to run its experiments. This approach is very ambitious, but it may require several years to be deployed, and it may turn out to be rater costly. OpenFlow is largely protocol independent and readily available on currently running networks. This justifies the interest of the research community for this approach.

Since OpenFlow is a relatively new technology currently under deployment, several research issues should be solved: among others, robustness, ease of management, performance and scalability. In this paper we focus on the OpenFlow data plane performance. We compare OpenFlow with Ethernet switching and IP routing functionalities in a Linux environment. Experiments were run in our Lab on a Linux PC exploiting an HW traffic generator to stress packet forwarding capabilities of the PC under test. Besides absolute performance values, which clearly depend on the specific hardware and software architecture, OpenFlow showed good forwarding performance and fairness properties.

II. Brief Introduction to OpenFlow

We briefly outline the main characteristics of OpenFlow. A more detailed and exhaustive documentation is available in the OpenFlow white paper [3] and in the OpenFlow specification [4].

The OpenFlow network architecture comprises several OpenFlow switches, one (or more) controller, a secure channel that interconnects the switch with the controller, and the OpenFlow protocol for signaling between the switches and the controller.

A classical IP router or Ethernet switch runs both the data and the control plane. The data plane is in charge of packet forwarding, whereas routing decisions are taken in the control plane. OpenFlow separates data and control plane functions: packet forwarding is executed in the OpenFlow switch on the basis of a flow table, whereas high-level routing decisions are moved to a separate OpenFlow controller. A proprietary protocol defines the messages exchanged between an OpenFlow switch and an OpenFlow controller: messages include information on received packets, sent packets, forwarding table modifications, statistics collection, etc. When the switch receives a packet that does not match any entry in the flow table, it sends the packet to the controller. The controller may drop the packet or may add a flow entry in the switch flow table, to force the switch to forward packets belonging to the same flow on a given path.

Each entry in the flow table of an OpenFlow switch contains a set of packet header fields, an action to be performed on packets matching the header field, and flow statistics. The

978-1-4244-6404-3/10/$26.00 ©2010 IEEE

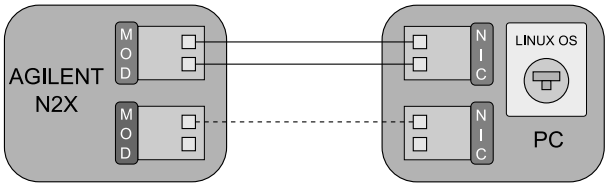

Fig. 1. Test Setup

packet header field includes: port id; VLAN tag; Ethernet type, source and destination address; IP protocol type, source and destination address; UDP/TCP source and destination port. Examples of possible actions are: send the packet to output port i, modify packet header field f or drop the packet. Flow statistics include number of packets and bytes for each flow, and the time since the last packet matched the flow (to ease the task of inactive flow removal).

Two types of OpenFlow switch can be identified. The first one is the hardware-based commercial switch, which typically uses TCAMs and a proprietary operating system to implement the flow table and the OpenFlow protocol. The second one is the software-based switch that use UNIX/Linux systems to implement the OpenFlow switch functions. Two flow tables are managed in the software based OpenFlow switch. The first one is named linear table and exploits wildcards in the packet header fields to match packets to flows. This table is of small size: it contains only 100 entries. The second table is an exact match table, exploiting a hashing function, and contains up to 131072 entries.

III. Experimental Setup

The experimental setup is shown in Fig. 1. The test machine (TM) is a standard PC mounting a C2SBX mainboard from Supermicro, an Intel Core2 Duo processor and 8 Gbyte of DDR3 1066 MHz RAM. Network connectivity is provided by two Intel PRO/1000 PT dual port 1 Gbit/s NICs plugged into PCI-Express bus. The operating system is Linux Ubuntu 8.10, kernel version 2.6.27, with bridging and NAPI enabled.

Each test is based on synthetic traffic generated using an Agilent N2X router tester [5] equipped with Gigabit Ethernet modules. The use of the router tester ensures both bottleneck-free traffic generation and high measurement precision not achievable with standard PCs and software traffic generators. Results are averaged over three runs lasting 60 seconds each.

Two port pairs are used (except for the fairness tests) to interconnect directly the TM and the router tester. Traffic is sent to the TM over the first link and collected over the second one. In the case of the fairness tests, the two links are both used to send traffic to the TM, and a third link (the dashed one in Fig. 1) is used to sink the traffic. All links use UTP cat 5e cabling.

Switching tests use directly the Linux kernel implementation of a 802.1d bridge. The bridge-utils software suite is used to create a virtual switch and to connect the test interfaces to it. Routing is performed using the IP forwarding engine available in the Linux kernel. OpenFlow tests use version 0.8.9r2.

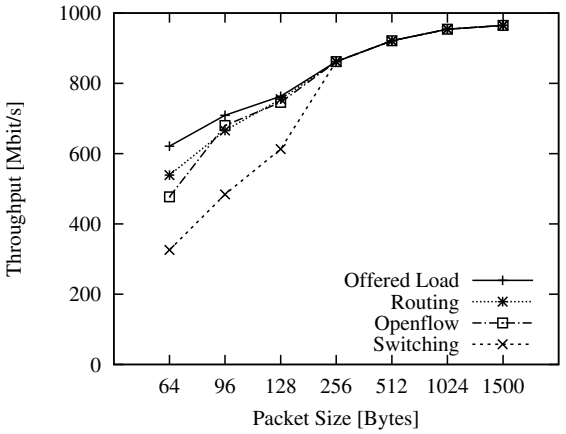

Fig. 2. Maximum single flow throughput versus packet size.

IV. Results

We present the main results of the experiments run in our lab to compare the performance of OpenFlow to the ones achievable by standard L-2 and L-3 forwarding. We mainly focus on achieved throughput as the main comparison metric, although we discuss packet latency in some scenarios.

A. Single-flow Performance

We start comparing the three technologies assessing the maximum throughput as a function of the packet size in Fig. 2, with single-entry forwarding tables. Using a single flow permits to assess the maximum throughput, because it minimizes the table lookup cost and it benefits from caching techniques. The offered input load is also plotted as a reference.

As expected, in all three cases, performance increases with packet size. This is due to the well-known fact that the operations performed to forward packets scale with the number of packets. Thus, the smaller the packet size, the higher the load on the CPU, which becomes the bottleneck. Both routing and OpenFlow perform well, being able to almost match the offered load, if not for small packet size. Having a single flow means that IP routing exploits the internal (hash-based) cache, which simply tests the IP header without the need to perform a demanding longest-prefix look-up operation. Thus, the small difference between OpenFlow and routing performance can be credited to the fact that the OpenFlow switch has to consider additional header fields with respect to the L-3 forwarder. Strangely enough, switching performs quite worse than the other two forwarding techniques. We could not find any explanation for this, apart from the suspicion that the bridging code is not optimized. However, starting from 256-byte packets, all three forwarding techniques are able to achieve the full wire speed.

Since small packets are the worst case, we take a slightly deeper look at this scenario. Fig. 3 and Fig. 4 report throughput and latency for 64-byte packets for different input loads. As expected, thanks to NAPI, the throughput reaches a saturation level and does not suffer from performance drops after the peak has been reached. From the latency plot, it is easy

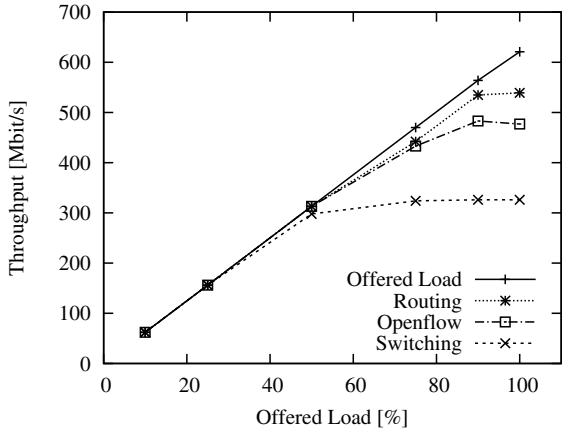

Fig. 3. Single 64-byte packet flow throughput.

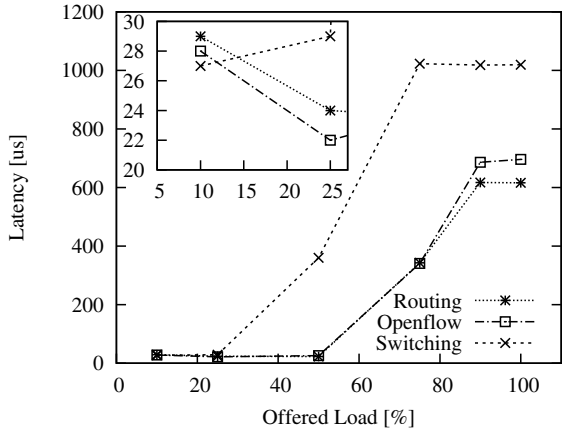

Fig. 4. Single 64-byte packet flow latency.

to discern the underloaded from the overloaded region. The results are very similar for all three technologies. In the underloaded region, the average latencies are quite small (slightly less then 30 μs).

B. Multi-flow Performance

We consider now the effects of the forwarding table size. We perform tests with three different table size: 8K , 64K and 128K, labelled in Fig. 5 as (s)mall, (m)edium and (l)arge. The large table is set to 128K entries because this is the maximum table size supported by the OpenFlow software.

To fill up the forwarding tables, we use bash scripts. The routing table is filled with C-class networks, while the OpenFlow rules are created using both different UDP source and destination port numbers. An extra note is needed for the switching table. This table is usually created automatically adding entries through the backward learning algorithm: entries are removed after a timeout expiration. Hence, no explicit commands to manipulate the table content are provided. To fill up the table, we set the timeout value to one hour, and, before starting any test, we send through the switch a proper number of packets with different source addresses.

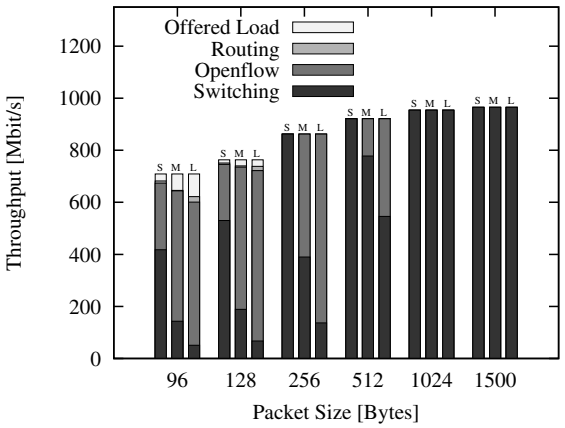

Fig. 5. Maximum throughput for different packet and forwarding table size. Each bar within the groups of three refers to different table size: (s)mall 8K, (m)edium 64K and (l)arge 128K entries.

To minimize the beneficial effects of caching techniques, which would make the comparison more difficult, we generate packets with random header fields (uniform distribution within the admissible range). More precisely, we use packets with randomly generated destination IP addresses for the routing tests, UDP port numbers for OpenFlow tests and destination MAC addresses for switching.

Fig. 5 presents the results. For each packet size, three columns are plotted: one for each table size. Furthermore, in each column the offered load and throughput achieved by the three forwarding engines are superimposed: offered load is plotted first, in the background, then, moving to the foreground, routing, OpenFlow and switching.

As expected, the larger the size of the forwarding table, the worse the achieved performance. However, while for both routing and OpenFlow the performance degradation is quite limited, the effect on switching is disastrous. For 96-byte packets, routing and OpenFlow present a relative performance drop of 0.1%, 5.4%, 8.9% and 0.6%, 5.0%, 11.3% respectively, for the three table size. Switching instead looses an astonishing 13.6%, 70.4%, 89.5%, with a throughput smaller than 10% of the link capacity in the worst case. Of course, increasing the packet size helps. Routing and OpenFlow need at least 256 byte packets to achieve wire-speed, for small table size. In the case of switching, the minimum packet size to obtain wire speed becomes 1024 bytes.

C. OpenFlow Table Types: Hash vs Linear

Up to now, we considered only exact matches for the OpenFlow forwarding rules. However, OpenFlow permits to set up rules with wildcards. The rules for exact matching are stored in a hash table, while the wild-card table is implemented using a linear search table allowing for up to 100 entries. The two tables are organized in a chain, and priority is given to the hash table matching.

Fig. 6 reports throughput performance when using the two tables for different packet size. Despite the small table size,

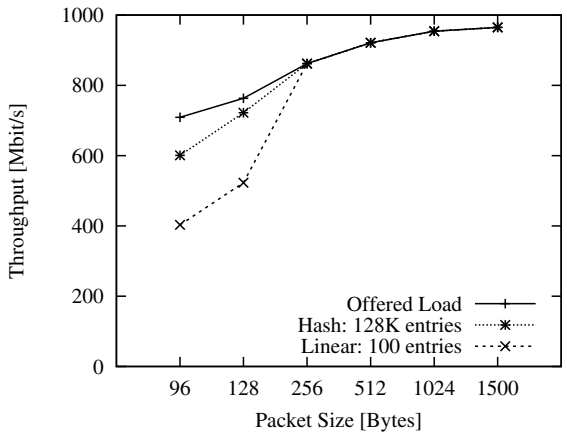

Fig. 6. OpenFlow throughput versus packet size with different table

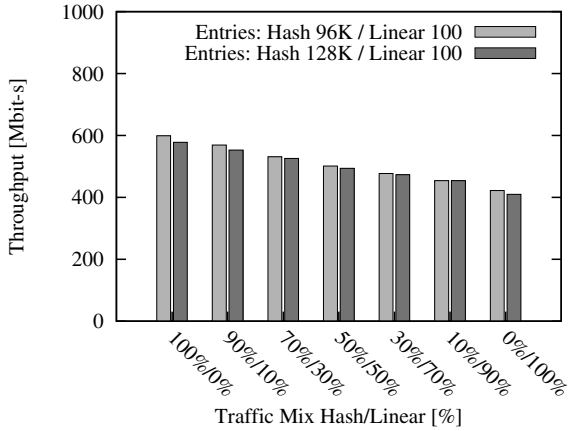

Fig. 7. OpenFlow throughput when varying the traffic mix for hash and linear tables.

the throughput achieved, when using the linear table only, is lower than the throughput of the hash table. The performance drop for 96-byte packets is about 33%.

Fig. 7 shows that mixing up traffic of flows that are forwarded using both tables does not modify forwarding performance too much. As expected, performance is always bounded by those obtained when using, respectively, the hash table only and the linear table only. The performance drop is proportional to the amount of traffic matching the linear table. This is due to the fact that the table chain smartly puts the hash table before the linear table.

The hash table should find the match in constant time, independently of the table size, while the lookup operation within the linear table should be directly proportional to its size. To verify this, we tried to estimate the average per packet processing time T_p. To estimate T_p, we use the model shown in Fig. 8: the packet latency L is the sum of the transmission time T_t, the propagation delay D for both the forward and reverse direction plus T_p. Therefore:

$$T_p = L - 2(T_t + D) \quad (1)$$

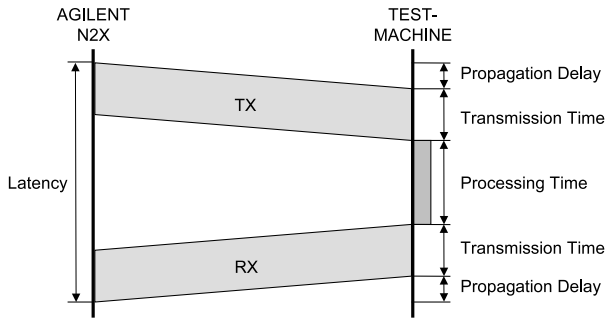

Fig. 8. Timing model to estimate the processing time from latency measurements.

L is measured by the router tester, T_t depends on the packet size S and the link capacity C ($T_t = \frac{S}{C}$) which are known. Since the UTP cables are short (3 meters), D is negligible.

Table I shows the computed processing times for different table size, table types and packet size. If we read the table by rows, the hash processing time is constant and independent of the table size. More surprisingly, also the linear table processing time seems to be constant and independent of the table size. This is in contrast with our expectations. Since the maximum table size is rather small, we believe that the effect of the table size is too small to be measurable if compared to other delay variations induced for example by CPU scheduling.

Reading the table by columns, we observe that the processing time is constant with respect to the packet size. This confirms two well-known facts: i) forwarding delay is a per packet operation, and ii) with small packets, performance is limited by the CPU.

D. Fairness

The fairness test uses two different flows. The first flow $F1$ has a fixed rate equal to 10% of the link capacity. The second flow $F2$ has a variable input load (40%, 50%, 60% and 90% of link capacity), and is used to overload the system. Both flows use 96-byte packets, and the forwarding table has 64K entries.

We measured fairness looking at the loss probability P_L for the two flows. In particular, we consider as a fairness parameter the difference between the two loss probabilities ($P_{L_{F1}} - P_{L_{F2}}$). If the difference is zero, the same relative amount of packets is lost by both flows, and the system is fairly managing the flows. If the difference is positive/negative, F1/F2 is obtaining more than its fair share.

Furthermore, we considered two different scenarios. In the first one, one input port is used for both flows, the same setup used for all other tests. In the second one, the two flows arrive at different input ports. In both cases, a free output port is connected to the traffic generator to collect statistics.

Results for the first scenario are reported in Fig. 9. We observe that all three forwarding technologies are quite fair: the packet loss difference is always smaller than 1.5%. Within

	Linear Table Size					Hash Table Size				
Packet Size [byte]	1	25	50	75	100	1	32K	64K	96K	128K
96	28,18	28,18	28,18	28,18	28,18	28,18	28,18	28,18	28,18	28,18
128	28,66	28,66	28,66	28,66	27,66	28,66	28,66	28,66	28,66	28,66
256	29,62	29,62	28,62	28,62	28,62	29,62	29,62	29,62	29,62	29,62
512	28,52	28,52	28,52	28,52	28,52	28,52	28,52	28,52	28,52	28,52
1024	28,33	29,33	29,33	29,33	29,33	28,33	28,33	28,33	29,33	29,33
1500	25,71	25,71	25,71	26,71	26,71	25,71	26,71	26,71	26,71	26,71

TABLE I
ESTIMATED OPENFLOW PACKET PROCESSING TIMES [μS] FOR DIFFERENT TABLE TYPES AND SIZE.

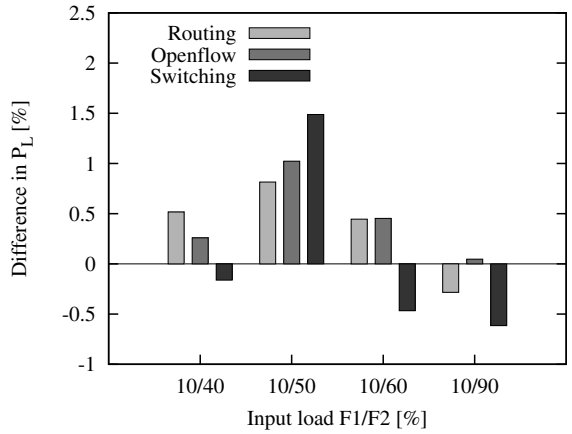

Fig. 9. Fairness test when flows are generated on a single link, for a packet size of 96 byte.

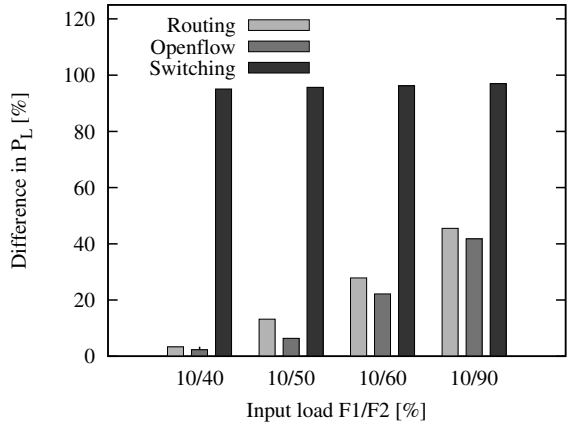

Fig. 10. Fairness test when flows are generated on two different links, for a packet size of 96 byte.

this small range, OpenFlow seems to do slightly better and switching slightly worse.

For the second scenario, the results are quite different (Fig. 10): flow F1 is given higher priority. The reason is the NAPI deficit round robin scheduler used within the Linux kernel, that gives service to the different network interfaces. The deficit round robin scheduler ensures a max-min fairness mechanism, which privileges the smaller flow (F1 in our case). The more the system is overloaded, the more this effect is evident. Both routing and OpenFlow clearly show this behaviour, and OpenFlow is slightly fairer. Instead, switching is already overloaded even at the lowest load. Having reached a saturation point, no difference can be observed by further increasing the load of the second flow.

V. CONCLUSIONS

The OpenFlow implementation in Linux systems is able to offer very good performance. The hash table performance matches almost always the ones obtained by the layer-3 routing. Only when the system is overloaded, e.g., small packets load the switch, throughput performance becomes slightly worse. Roughly a performance drop of 11% is measured for 64-byte packets only, which reduces to 3.5% for 96-byte packets and 128K forwarding table entries (hash table). When using the linear table (wildcards), performance becomes worse than those of the hash-based case, but wire speed is reached with packets larger or equal to 256-byte. Finally, OpenFlow shows good fairness capability in dealing with multiple flows. In summary, the ability to handle up to 128K exact matches and up to 100 wild-card matches, combined with good throughput, latency and fairness performance, seems an encouraging result toward the use of OpenFlow in production networks.

Surprisingly, the Layer-2 switching Linux implementation shows a performance drop of about 40% and 30% respectively with respect to layer-3 routing and OpenFlow for single 64-byte packet flows. Furthermore, layer-2 switching suffers large forwarding tables: in the worst case of 96-byte packets and 128K table entries, layer-2 switching is unable to sustain an input load of 10%.

ACKNOWLEDGMENTS

These activities were developed in the framework of the FEDERICA project, funded by the European Commission.

REFERENCES

[1] GENI: Global Environment for Network Innovations. Web site: http://geni.net.
[2] FEDERICA: Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures http://www.fp7-federica.eu/.
[3] OpenFlow: Enabling Innovation in Campus Networks. Web site: http://www.OpenFlowswitch.org/documents/OpenFlow-wp-latest.pdf
[4] OpenFlow Switch Specification v0.8.9. Brandon Heller (brandonh@stanford.edu). Web site: http://www.OpenFlowswitch.org/documents/OpenFlow-spec-v0.8.9.pdf
[5] Agilent N2X router tester. Web site: http://advanced.comms.agilent.com/n2x/