OpenFlow driven ethernet traffic analysis

OpenFlow Driven Ethernet Traffic Analysis

Andrea Bianco*, Vengatanathan Krishnamoorthi†, Nanfang Li* Luca Giraudo‡
*Dipartimento di Elettronica e delle Telecomunicazioni, Politecnico di Torino, Italy,
†Department of Computer and Information Science Linköping university, Sweden,
‡VMware, Inc, USA.
Email: *{firstname.lastname}@polito.it, †{firstname.lastname}@liu.se ‡{lgiraudo}@vmware.com

***Abstract*—Software Defined Networking (SDN) is a new networking paradigm that permits to slice network infrastructures. An example of SDN is the OpenFlow framework, where the control plane runs on a separate device, called controller, that manages data forwarding switches. The OpenFlow protocol ensures communications between OpenFlow switches and the OpenFlow controller. Before widely deploying OpenFlow based networks, scalability and performance of such networks should be studied and better understood.**

In this paper, the scalability of NOX, one of the most popular OpenFlow controller, is analyzed through both simulation and lab measurements. We perform an Ethernet trace analysis on the controller by defining flow characteristics as would be seen by an OpenFlow controller. We study the potential trace impact on an OpenFlow controller, analyzing among others, the number of flows, flow inter arrival times, traffic volumes and flow size distribution. Our results permit to discuss the feasibility of running OpenFlow networks with a single commodity PC as the controller in a mid-size campus network.

I. Introduction

Production networks must provide a high degree of availability and security, because networks are a critical component of today's world. At the same time running research experiments on realistic networks is fundamental to test new features for future networks, but innovative network protocols and architectures barely find a chance to be tested on a production network. These two trends are often in contrast on real networks. To avoid modifying the installed base, several new network protocols have been introduced to meet the new demands, leading to a complex and ossified network architecture.

To cope with this problem, projects like GENI [1] and FEDERICA [2] propose to slice network resources through network virtualization. In such architectures, it becomes possible to run experiments on a potentially large scale network within a given network slice without interfering with other slices. These solutions are promising but cannot be readily used today in production networks and require considerable investments.

Fairly recently, OpenFlow has been proposed in the framework of the 4D network architecture [3]. OpenFlow separates the control plane (Decision, Dissemination) and data plane (Discovery, Data) to greatly simplifying network management. Furthermore, administrators run all management and control functions in a single device, the OpenFlow controller, it would be possible to slice the architecture into a production network and several experimental networks with very limited interaction among them. These benefits are mainly due to the flow-based traffic switching/forwarding mechanism and to the centralized control paradigm. The OpenFlow controller has a global view of the network and configures the switches to forward packets on the basis of a specified flow definition. All control and management functionalities, e.g., routing decision, QoS management, flow identification, are performed in the controller. The OpenFlow switch simply acts as a forwarding engine that operates according to the rules set up in the forwarding flow-based table(s) by the OpenFlow controller.

Before widely deploying an OpenFlow network, performance and scalability features must be carefully studied. The most critical component is the centralized OpenFlow controller. In this paper we choose NOX [4], one of the most popular OpenFlow controller, and we study its performance through simulation and in-lab measurements. Furthermore, we analyze traffic traces captured on the Politecnico di Torino Ethernet-based campus network. More precisely, we extract packet headers to determine flow behaviour as if the captured traffic was running in an OpenFlow based infrastructure, to study the flow arrival load in an OpenFlow controller that manages the OpenFlow network. The analysis permits to identify the average and worst case loads on the OpenFlow controller. Flow size distribution, flow inter-arrival times and traffic volumes are also derived.

II. Related work

Many SDN controllers exists nowadays, for instance, NEC proposed the ProgrammableFlow controller [5], Big Switch Network has the Big Network Controller [6] and Cisco developed the XNC (Extensible Network Controller) [7]. These are all proprietary solutions. In the meantime, open souce controllers like Beacon [8], Floodlight [9], Nox/Pox [4] and Flowvisor [10] are quit popular as well. Given the large amount of SDN controllers, the benchmarking of their performance could be very important for selection. To the best of our knowledge, Few works related to OpenFlow controller performance and benchmarking exist in the literature. We briefly discuss the main differences of the existing works with respect to our approach. OpenFlow wiki page [11] presents some benchmark results obtained by using cbench [12] for three different OpenFlow controller implementations, namely NOX, Beacon and Maestro. The results were obtained by exploiting the boost libraries and Google's thread-caching malloc features. Our work focus on the performance provided by off-

the-shell PC without any hardware and software optimization. In [13] the authors have presented an open software framework which permits to test the OpenFlow-enabled switches. The main conclusion of this work is that it is essential to consider the data plane and the control plane traffic together for accurate performance evaluation, rather than only concentrate on the control plane traffic. During our experiments and trace analysis, we followed their suggestion and combined the control and data plane traffic together to evaluate the controller performance. Jarschel et al. [14] presented a model for estimating the forwarding speed and blocking probability of an hardware based OpenFlow switch. By using such a model, it is possible to estimate the switch performance under any given traffic input. It would be interesting to extend their work to support software OpenFlow switch to compare our experimental results with their analytical results, we leave this as future work. Finally, NOX-MT [15], an optimized multi-threaded NOX contorller targeting on multicore server has been developed, the performance results show that it can efficiently utilize additional computation resource from many core server and it outperforms all the existing controllers.

III. OpenFlow Controller Performance

In this section we analyze one of the OpenFlow controllers, named NOX, and highlight its performance limitations, which will be used to evaluate OpenFlow network scalability using the captured traces in following sections. NOX is one of the most common OpenFlow controllers, originally developed at Nicira Networks. It became public in 2008 and, since then, many researchers integrated different applications into NOX, making it even more popular. We cloned the NOX "destiny" branch from github because this version is multi-threaded, with optimized code. We tested NOX's performance using first cbench [12] and later via in-lab measurements. The controller is hosted on a commodity off-the-shelf PC equipped with an Intel Xeon dual-core processor at 3Ghz, the RAM is 8GBs (667Mhz DDR2 SDRAM). All Ethernet interfaces are gigabit capable and the OS is 32bit Ubuntu 9.10 with kernel version 2.6.31-14.

A. Cbench Simulation Test

Cbench emulates an OpenFlow network with a variable number of switches and end hosts. All switches are OpenFlow enabled, and send new flow requests to the controller and record response statistics. Cbench supports two operational modes: latency mode and throughput mode. In latency mode, cbench sends one request (i.e, packet_in) to the controller and waits for the response (i.e, flow_mod) before sending the next request. In throughput mode, cbench tries to send many requests simultaneously so as to find the maximum controller throughput. More precisely, cbench adapts the request rate to reach a balance, i.e, $\#packet\_in = \#flow\_mod.$ which defines the controller's throughput.

The server running the NOX controller is a Dell server equipped with an Intel Xeon dual core running at 3.0GHz.

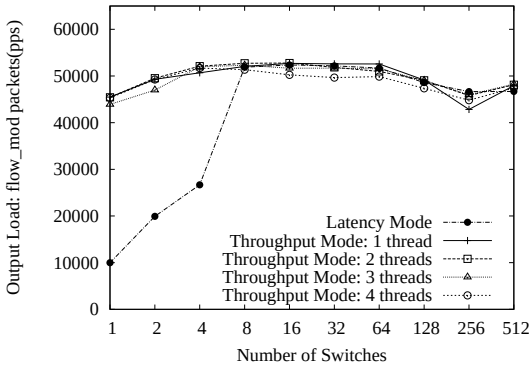

Fig. 1: Openflow controller NOX performance test under cbench as the number of switch increases, in latency mode and in throughput mode

Fig. 1 shows the NOX throughput for the switching application[1] as a function of the network size, i.e, the number of switches connected to it. When cbench runs in latency mode, few switches are not able to saturate the controller. As soon as the number of switches exceeds 8, the response from the controller becomes stable, around 52000 packets per second (pps). On the other hand, cbench running at throughput mode easily saturates the controller, even with a single switch in the network. Increasing the number of switches does not significantly modify controller's performance. When running NOX in multi-thread mode, throughput improvement is very limited. The curves in latency mode and throughput mode are very similar, and, in both cases, the controller can forward 52000pps. Thus, a commodity server running NOX OpenFlow controller could handle roughly 52000 flows per second.

B. In-lab Experimental Test

Although the controller performs similarly during the test with cbench in different modes, the software based simulation approach could be misleading and the accuracy of such an approach is always open to question. To confirm our results, we built a testbed in our lab, as shown in Fig. 2. PC1 is the same server as the previous one and it runs the NOX controller, while PC2-PC5 are standard servers running Open VSwitch (OVS) [16], a software implementation of the OpenFlow switch. The hardware traffic generator and receivers are Agilent N2X RouterTester [17] with multiple Gigabyte modules. During our experiments, we activate each switch and check if the bottleneck is the switch or the controller. If the switch is the bottleneck, we incrementally add more switches to saturate the controller.

As shown in Fig. 3, since one switch can handle at most 22000pps, based on the previous evaluation we know that this volume is too low to saturate the NOX controller. As the input load increases, the switch starts to drop packets, thus further

[1] Preliminary results show that NOX switching and routing applications perform very similarly in terms of flow handling capabilities. Thus, we stick to NOX switching in this section.

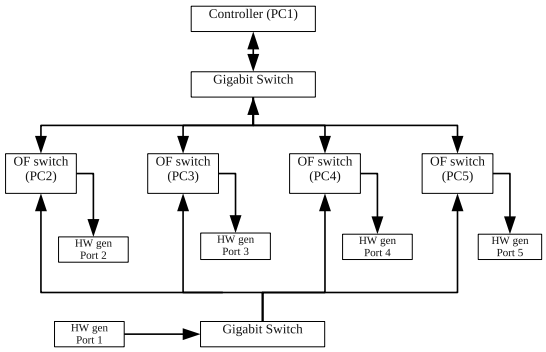

Fig. 2: The hardware topology of Openflow controller NOX performance test with in-lab experiment

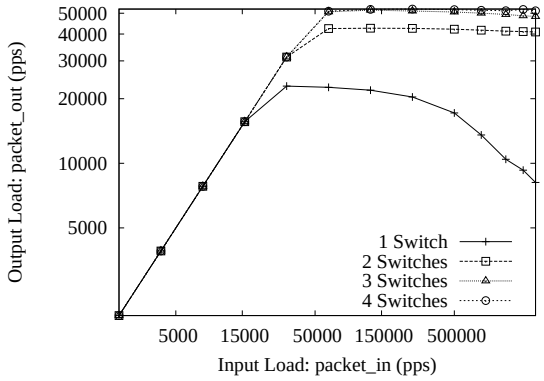

Fig. 3: The results of Openflow controller NOX performance test with in-lab experiment

reducing the controller load. This could be mitigated with more switches being active simultaneously. When activating the 2nd, 3rd and finally the 4th switches, controller loads over 52000pps could be successfully obtained. The CPU loads on the NOX server and switches were also checked to confirm that the bottleneck is at the controller and not at the switches.

The experimental results obtained from the cbench and in-lab measurements are coherent. In summary, a commodity PC running the NOX OpenFlow controller can handle around 52000 flows per second. In the remainder of this paper, we capture and analyze traffic traces to understand if the NOX OpenFlow controller is able to handle flows generated in a middle sized campus network.

IV. Traffic Analysis Methodology

We wish to cross validate the controller performance with Ethernet based traces. Indeed, our objective is to study OpenFlow flow characteristics, which are fundamental to the controller. We are not running an OpenFlow network, but we captured the campus network traffic and extracted information valuable from an OpenFlow network's point of view. Although we captured traffic on a conventional Ethernet network, packets can be analyzed in terms of OpenFlow applications running in the controller. i.e., identifying flows

	Duration	Packets	Size
External	24 hours	156,984,191	13.78 GB
Internal	24 hours	138,286,565	13.78 GB

TABLE I: Trace Characteristics

for different OpenFlow control applications. We choose three different control applications to study:

1) **OpenFlow**: Switching based on all packet header fields defined in OpenFlow specification [18];
2) **Layer234**: Switching based on source/destination MAC, IP addresses and TCP/UDP port numbers;
3) **Layer23**: Switching based on source/destination MAC and IP addresses.

We assume that there is a single controller to handle all the traffic inside the campus network, and that all networking nodes are OpenFlow enabled devices.

The Politecnico di Torino campus backbone network is based on a multi-root tree topology. Each switch in a Department is connected to two backbone core routers for redundancy; the remaining switching nodes inside the Department form a tree.

The traces used for the analysis were taken from two links of the DET (Dipartimento di Elettronica e delle Telecomunicazioni), one inside the department (named internal link) and the other one connecting the backbone and the department gateway (named external link). There are roughly 1000 active hosts in the department and more than 5000 hosts in the campus network. We captured packets for a week, from January 23rd until January 29th 2012, with TCPDUMP from a standard Linux PC. Only packet header fields were recorded. More than 60GB binary data in pcap format for each link have been captured. To process data, we exploit libtrace [19] to get target header fields according to different control applications. The time-stamp for each packet has also been analyzed to derive flow inter-arrival time information. Several ad hoc scripts were used to extract interesting parameters from the traces.

V. Trace Analysis

We present the main results obtained by analyzing the traces considering different OpenFlow control applications. All the results fix the Flow Inactivity Time (FIT) parameter at 5s, i.e., if no packet belonging to a given flow is received for more than 5s, the corresponding flow entry is removed from the switches' flow table, except the results directly measuring the FIT value impact on controller. The 5s value was obtained by the source code inspection of the NOX [4] controller.

	Control application	Number of flows	Avg flows/s
External	OpenFlow	9693169	112.19
	Layer234	9672603	111.95
	Layer23	6347418	73.47
Internal	OpenFlow	8034403	92.99
	Layer234	8018252	92.80
	Layer23	5342101	61.83

TABLE II: Overall Statistics without the Scan

A. Daily Trace Statistics: Scanning Detection

Due to the lack of space, we only report the main results obtained by analyzing the large data set. We selected the 24 hours trace from Wed 9:30am until Thursday 9:30am to perform the daily analysis.

Tab. I and II report the aggregate statistics. We run the same statistical analysis by considering different time scales: 60min, 30min, 5min and 5s respectively. The results are shown in Figs. 4 and 5. We only include graphs related to 5min time scale. Fig. 4 reports results for the external link traffic. Quite surprisingly, we discovered that the number of flows varies significantly during the day. The maximum number of flows is almost 10 times larger than the minimum one, and there is no evident day-night behavior for the number of flows. The same phenomenon is observed for the flow inter-arrival time, but not for the traffic volumes. This effect is not seen if the traces are analyzed over 30 minutes or 60 minutes intervals, but only for the 5min and 5s intervals. Since the traffic volume is stable, we suspected that there could be some periodic scanning applications running in our campus network. After a deeper analysis through Tstat [20], we found that, once every 15min, a server performs a scanning with nmap to detect the liveliness of all the hosts in the campus network. The scanning depends on the PoliSave project, running at Politecnico di Torino, that tries to turn off unused host for power saving purpose. A nmap daemon is used to scan all IPs to record servers' liveliness every 15 minutes. The scanning service translates to a large number of flows with low volumes (i.e, each packet belongs to a different flow). This would cause undesirable and excessive OpenFlow control traffic between switches and the controller, because each packet should be sent to the controller, each flow being one packet long. In this case, packet based switching, being distributed in switches and not centralized in the controller, would be more efficient than flow based switching. This type of traffic if not properly managed may lead to OpenFlow controller overload.

B. Daily Trace Statistics: External vs Internal

Results for the traffic after filtering out the scan traffic are shown in Fig. 5. We did not report the plots for the three applications but only leave the *OpenFlow* as the target application to highlight the statistics on two different links.

As expected, the captured traffic exhibits a strongly periodic day-night pattern as shown in fig. 5. From left to right, each figure represents respectively, the total number of flows, the flow inter arrival times and the traffic volume. Each point is the average over 5 minutes. The daily number of flows is 2-3 times larger than the number of flows at night, with the exception of Saturday and Sunday because fewer flows are observed thus reducing the difference between day and night. The flow inter arrival time roughly behaves inversely proportional to the flow number. During the day, shorter flow inter arrival times are observed compared to the night time. The minimum and maximum inter arrival times differ roughly by a factor of 2-3, similarly to the case of number of flow. For the traffic volumes, besides the day-night pattern, more outliers can be observed. No direct relation can be found between the number of flows (or the flow inter arrival times) and the traffic volume spike. This means that OpenFlow can handle elephant flows (high volume flows) efficiently, because only the first packet belonging to each flow is sent to the controller and all the subsequent packets follow the same path without any interaction with the controller. Thus, whereas volume spikes can be observed, the number of flow does not directly depend on the volume.

Comparing the two different links, the external one has more flows and a larger traffic volumes, as expected, because it aggregates all the traffic within DET. It is interesting to notice that the shape of the two curves are similar. This suggests that: i) the traces we captured share a large amount of data, i.e, the packets sent to the external world (public Internet or other Department buildings) traverse both links; ii) the internal traffic, i.e, traffic appearing only on the internal links, is very limited.

Link/Flow Definition	Duration	Average	Min	Max
External/ *OpenFlow* or *Layer234* or *Layer23*	1 hour	4366.8	1218.3	11396
	30min	2183.4	538.39	8006.2
	5min	363.9	62.87	2810.5
	5sec	6.065	0.097	115.31
Internal/ *OpenFlow* or *Layer234* or *Layer23*	1 hour	3658.3	1064.0	9507.8
	30min	1829.1	491.11	6042.7
	5min	304.86	60.11	2070.9
	5sec	5.081	0.084	105.79

TABLE III: Flow Volumes Statistics in MB

C. Daily Trace Statistics: Worst Case Analysis

We focus now on minimum and maximum values of measured parameters to highlight the worst and best operating conditions for the controller over different time intervals. Fig. 6 shows the number of flows and the flow inter arrival times for the external link over different time intervals. Obviously, the number of flows decreases as the time scale decreases. Furthermore, the gap between the maximum and minimum flow numbers increases as the time scale decreases. For instance, in a one hour interval the difference between the minimum and maximum number of flows is roughly 3 times, whereas it becomes 15 times over a 5s interval. The same result holds for the flow inter arrival times. The average flow inter arrival times show a slight discrepancy when observed over different time scales, because the average on short time intervals is different from the long term average due to the border effect. We omit the results for the internal link because they show the same statistical behavior as the external one.

Tab. III presents the traffic volume statistics for different time intervals. As expected, the traffic volume scales down as the time interval decreases. Traffic volumes show a larger variation compared to the number of flows if comparing the maximum and minimum values. This is justified by the existence of mice and elephant flows.

In summary, the worst case traffic is quite different from the average one, especially when considering small time intervals

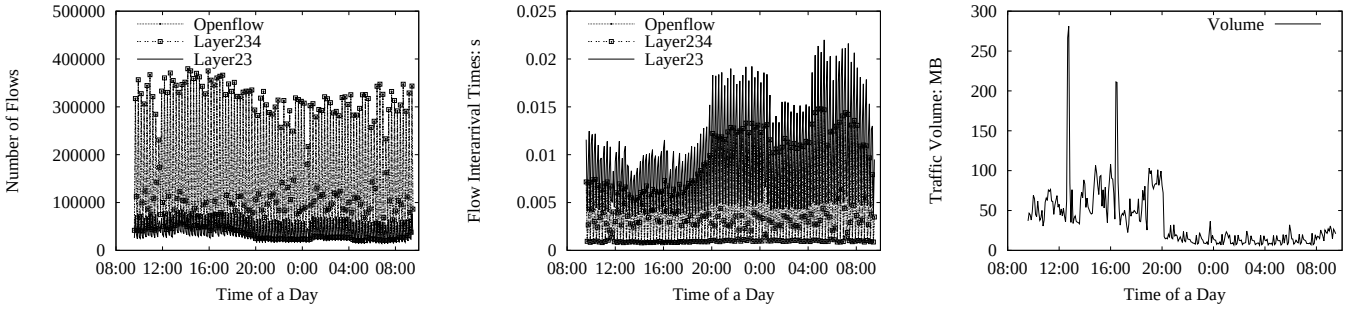

Fig. 4: Daily statistics on the external link averaged over 5min without filtering for different control applications. From left to right: number of flows, flow inter arrival time, traffic volume.

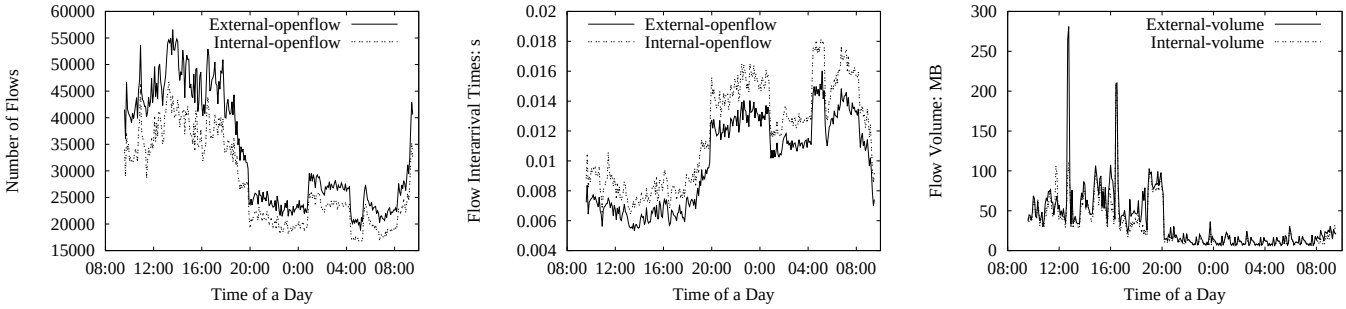

Fig. 5: Daily statistics on the external and internal links averaged over 5min with filtering. From left to right: number of flows, flow inter arrival time, traffic volume.

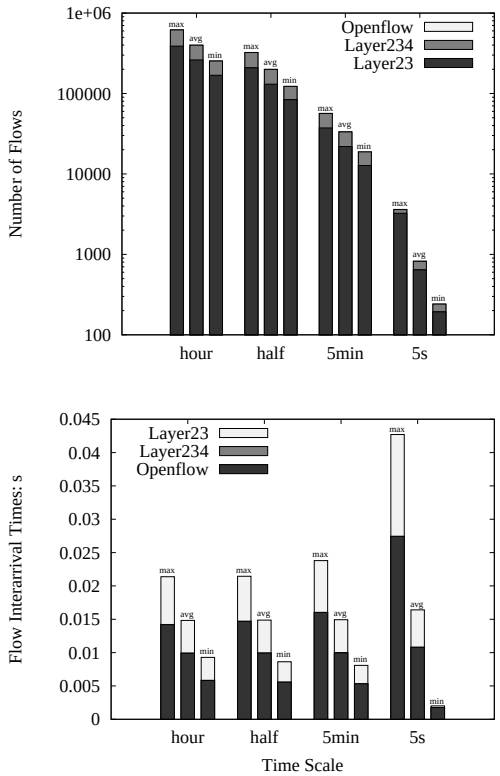

Fig. 6: External link capture. Maximum, average and minimum values for the number of flows (top) and flow inter arrival time (bottom) for different time intervals.

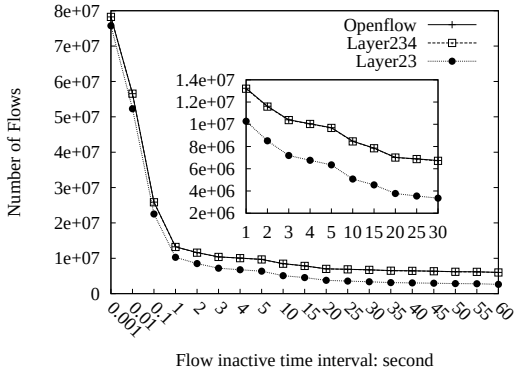

Fig. 7: Number of flows on the external link for different flow inactivity times.

(i.e, 5s, 5min). For instance, in Fig. 6, the maximum number of flows is 6 times larger when compared to the average value for a 5s interval. This must be taken into account when designing OpenFlow network architectures because the capability of OpenFlow controller should be several times larger than the expected average load to avoid traffic losses or large flow forwarding delays.

D. Flow Inactivity Parameter and Flow Size Distribution

In this subsection we show how the flow inactivity time (FIT) influences controller performance. Fig. 7 shows the number of flows for the three applications with different FIT

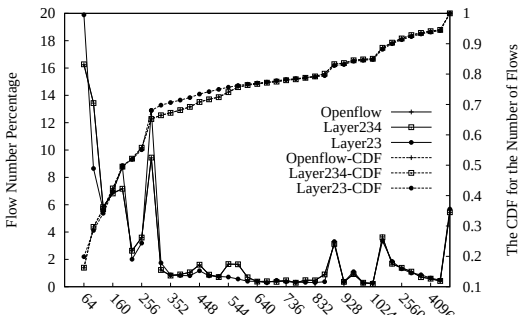

Fig. 8: Flow size distributions for different control applications for traces captured on the external link.

values, for traces captured on the external link. The number of flows increases as the FIT value decreases. When the FIT is less than 1s, the curve is very sharp, whereas when the FIT value exceeds 1s, the curve becomes smoother. Results suggest that OpenFlow switches should be designed with the FIT larger than 1s to keep low control traffic exchanged with the controller. If the flow table size is large enough, larger FIT values would be desirable.

Fig. 8 report the distributions of the number of flows with respect to the flow size. More than 70% of flows have a size smaller than 352 Bytes, whereas more than 20% of flows are larger than 832 Bytes, and 6% of flows have a size larger than 4096 Bytes. This result means that OpenFlow controllers deal with small flows for more than half of the time. This reduces the advantage of flow based forwarding with respect to packet by packet forwarding. In this situation, a high volume of control traffic is created and the relatively small number of packets belonging to a flow can not compensate the initialization phase needed to set up a flow entry in switches.

VI. Conclusions

We studied the performance of the OpenFlow controller NOX with simulation and experimental tests. We showed that OpenFlow controller running in a commodity server can handle roughly at most 52000 flows per second. Then, we selected three OpenFlow control applications and we analyzed OpenFlow traffic statistics on traces captured on our campus network, focusing on two links at the DET. The number of flows in this mid-sized network is not large, around 110 flows/s on average, but ranging to 60000 flows/s in the worst case. Simulation and lab measurements show that one OpenFlow controller running in commodity server could manage this number of flows, at least for the considered control applications. We also showed that traffic volumes are not related to the number of flows, owing to the contribution of elephant flows.

However, results related to the scanning service and to the flow size distributions reveal that, in current networks, mice-flows are dominant. In this context the centralized flow based forwarding solution of OpenFlow becomes of little advantage with respect to the classical packet by packet distributed forwarding scheme in terms of performance. Indeed, many control messages would be generated and setting up flow entries becomes not so useful due to the limited number of packets belonging to the same flow.

Acknowledgments

We would like to thank Antonio Lantieri and Marcello Maggiora, members of the IT Infrastructure Division-Information Technology Area of Politecnico di Torino, for their support in the link mirroring configuration and Kamran Ali for processing some of the traces.

References

[1] "GENI: Global Environment for Network Innovations." [Online]. Available: http://geni.net

[2] P. Szegedi, J. Riera, J. Garcia-Espin, M. Hidell, P. Sjodin, P. Soderman, M. Ruffini, D. O'Mahony, A. Bianco, L. Giraudo, M. De Leon, G. Power, C. Cervello-Pastor, V. Lopez, and S. Naegele-Jackson, "Enabling future internet research: the FEDERICA case," *Communications Magazine, IEEE*, vol. 49, no. 7, pp. 54–61, 2011.

[3] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 41–54, October 2005.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 105–110, July 2008.

[5] "NEC ProgrammableFlow controller." [Online]. Available: http://www.necam.com/SDN/doc.cfm?t=PFlowController

[6] "Big Network Controller." [Online]. Available: http://www.bigswitch.com/products/SDN-Controller

[7] "Cisco XNC Extensible Network Controller." [Online]. Available: http://www.cisco.com/web/solutions/netsys/CiscoLive/XNC.pdf

[8] "The Beacon Openflow Controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13.

[9] "Floodlight Controller." [Online]. Available: http://www.projectfloodlight.org/floodlight

[10] "Flowvisor Controller." [Online]. Available: https://openflow.stanford.edu/display/DOCS/Flowvisor

[11] "OpenFlow Wiki page on controller performance comparisons," http://www.openflow.org/wk/index.php/Controller_Performance_Comparisons.

[12] "Cbench." [Online]. Available: http://www.openflow.org/wk/index.php/Oflops

[13] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: an open framework for openflow switch evaluation," in *Proceedings of the 13th international conference on Passive and Active Measurement*, ser. PAM'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 85–95. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28537-0_9

[14] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11. ITCP, 2011, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043468.2043470

[15] "On Controller Performance in Software-defined Networks," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'12.

[16] "Production Quality, Multilayer Open Virtual Switch." [Online]. Available: http://openvswitch.org

[17] "Agilent N2X router tester," http://advanced.comms.agilent.com/n2x/.

[18] "OpenFlow Switch Specification," Feb. 2011, version 1.1.0. [Online]. Available: http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

[19] S. Alcock, P. Lorier, and R. Nelson, "Libtrace: A Trace Capture and Processing Library," University of Waikato, Tech. Rep., May 2010.

[20] "Tstat: TCP STatistic and Analysis Tool." [Online]. Available: http://tstat.tlc.polito.it/index.shtml