Position paper: Reactive Logic in Software-Defined Networking: Accounting for the Limitations of the Switches

Roberto Bifulco, Maurizio Dusi
NEC Laboratories Europe
{firstname.lastname}@neclab.eu

***Abstract*—SDN provides an abstract view of the network in which the switches expose a homogeneous set of capabilities. Unfortunately, the abstraction process may hide important information about the devices that finally impacts on the network operations. Although SDN implementations, like OpenFlow, already account for some device specific properties, we believe that other important ones have been so far overlooked. To this regard, we focus on the switches' performance at handling control messages and on how this may affect the control plane design. To support our discussion, we provide theoretical examples and simulation-based experiments. We conclude that accounting for switches performance at handling control messages is a crucial step in the design of advanced SDN control plane features.**

I. Introduction

Software-Defined Networking is changing the way we think about networks, introducing new abstractions that simplify the network management, operations and the control plane design in general. One of the advantages in providing abstractions resides in the possibility of dealing with a homogeneous set of objects, which greatly ease the design of a system. However, abstraction does not come at no cost. Complex systems, such as a computers network, are composed of devices with specific capabilities and limitations, which the abstraction process may hide.

SDN is not immune to this issue. Taking as an example OpenFlow, the Controller is provided with a nice forwarding element (FE) abstraction, where each FE can host a number of forwarding entries that implement a *Match-Action* approach. That is, a FE, or switch in the OpenFlow terminology, can host in its flow table (FT) a set of flow table entries (FTE), each of which specifies a network flow (the Match part) and the operations to apply to it (the Action part). Still, switches can be very different from each other. For instance, OpenFlow is implemented in both software and hardware switches [7], [14]–[16], which have different capabilities and performance [6]. Although OpenFlow provides some details on the switches, such as the number of flow tables and their maximum size, properties like performance limits are hidden in the abstraction process. This paper provides a few examples to highlight what is the impact on the network operations when ignoring these hidden properties, and how the control plane may be designed to overcome some of the highlighted issues. Even if we focus on the OpenFlow case, we believe that our arguments are valid for any SDN incarnation, where an entity is in charge of deciding the packet forwarding behavior of a group of FEs.

We will deal in particular with some specific OpenFlow operations, that is, the handling of *packet_in* and *flow_mod* messages, which help in implementing a relevant part of the functions offered by OpenFlow. To help the reader in understanding the relevance of these messages, after introducing their operations, and summarizing the currently measured switches performance limitations, we list some applications in which these control messages are required (Section II). We then discuss how the performance of a switch at handling these messages can affect the network operations and, ultimately, how a controller could benefit from gathering additional knowledge about the switches to this regard (Section III). Since *packet_in*s and *flow_mod*s are usually strictly related to the characteristics of the network flows traversing the switches, we also present a simulation-based experiment. Using the experiment we justify the need to handle the switches limitations taking into account the dynamic behavior of the network traffic (Section IV). We finally provide an overview of possible solutions in Section V and our conclusions in Section VI.

In summary, with the help of both theoretical examples and simulations based on real data, our work aims at supporting the following statements:

- taking into account switches' limitations and differences can improve the network performance and/or reduce the switch resources requirements;
- the control plane needs to monitor the switch/controller interactions and combine this information with the knowledge of the switches limitations.

II. Interacting with the switches

In OpenFlow the ability to support advanced control plane features comes from the possibility of dynamically interacting with the switches. In this, the *flow_mod* and *packet_in* messages play a central role. This section briefly describes these messages, it provides a summary of OpenFlow switches performance limitations at handling them and a short overview about their usage in some applications.

978-1-4799-6919-7/14 $31.00 © 2014 IEEE
DOI 10.1109/EWSDN.2014.22

Switch model	flow_mod/sec	Information source
HP ProCurve 5406zl	275	Scientific paper [13]
HP ProCurve J9451A	40	Scientific paper [6]
Fulcrum Monaco Ref.	42	Scientific paper [6]
Quanta LB4G	38	Scientific paper [6]
OpenVSwitch	408	Scientific paper [6]
NoviSwitch 1248	1000	Company website [15]
NoviSwitch 1132	500	Company website [15]

TABLE I: Switches perfomance at handling *flow_mod*s

A. *flow_mod and packet_in*

A *flow_mod* message is sent from the OpenFlow controller (OFC) to an OpenFlow switch to install a FTE. Hence, it is the message that enables the OFC at programming the forwarding plane of the switch.

A *packet_in* message is sent from a switch to a controller when a packet is received on one of the switch's ports and one of the following conditions is true: (i) there is a FTE that matches the packet and whose action is "send to controller"; (ii) there is a flow table miss (i.e, there is no FTE matching the packet) and the default action in such a case is "send to controller". Using *packet_in* messages the controller has a possibility to visualize the network traffic flowing through any port of any switch in the network.

B. *Switches' performance limitations*

Some scientific papers report a performance issue in the rate of *flow_mod* per second different switches can handle. Mogul et al. [13] find that a HP ProCurve 5406zl can handle only 275 *flow_mod*/sec. Huang et al. [6] find that HW switches support around 40 *flow_mod* per second, while OVS (soft-switch) supports around 400/sec on their Xeon X3210.

The importance of the switches performance at handling the control messages is further highlighted by some companies. For instance, NoviFlow advertises its switches by presenting performance numbers in terms of *flow_mod*/sec. For their switches NoviSwitch1132 and NoviSwitch1248, they report respectively 500 and 1000 *flow_mod*/sec.

The results cited in this section, summarized in Table I, show the relevance of this limitations and that there is a significant difference in the performance offered by OpenFlow switches at handling *flow_mod*s. Even if we do not have data available on the *packet_in*s handling performance, it is expected a similar heterogeneous performance landscape.

C. *OpenFlow applications*

To help the reader in understanding why it is relevant to care about *packet_in*s and *flow_mod*s, we provide a brief overview of OpenFlow applications that are making use of them. In [5], [8], to achieve an optimal bandwidth utilization over the network links connecting datacenters, frequent dynamic reconfiguration of the network (sending *flow_mod* messages) are performed. *Packet_in* messages are used to locate end-hosts in the network and to authenticate network users in [1]. While the *flow_mod* messages are used to enforce anti-spoofing rules. In [1], [3] is provided the possibility to select waypoints in the packets' routes, to specify which path to use for a given network flow or to provide QoS constraints. All these on-demand operations are enabled by the dynamic sending of *flow_mod* messages. In [17] the handling of *flow_mod* messages is the most expensive operation to implement service function chaining[1]. Also, *packet_in* messages are used to correlate flows entering and exiting a given network function. In broadband access networks [18], *packet_in* messages are used to identify new subscribers' session initiation, in order to steer, by sending *flow_mod* messages, corresponding flows towards the middleboxes (e.g., BRASes) in charge of establishing the sessions. In [12], [19], *packet_in* messages are used to capture relevant information for running anomaly detection algorithms, while *flow_mod* messages are sent to install FTEs for the legitimate flows which don't require further inspection.

III. Exploiting the SDN centralized control

In the presented applications, the controller has no information about the ability of a given switch in generating *packet_in*s or in receiving *flow_mod*s. In this Section we introduce a simple theoretical example that shows the disadvantages of not knowing this information. Using this example as a starting point, we show how the centralized view of the network can exploit the knowledge of switches performance limitations. Finally, we introduce a way to compute the theoretical network limit in handling control messages, if the control plane takes into account the switches limitations.

A. *A theoretical example*

In Figure 1 we show a network composed of six switches. Each switch has a limited number of resources, whose occupation of is represented by a number on top of the switch. For instance, $30/100$ indicates that the switch is handling 30 *flow_mod*/sec out of a maximum ability to handle 100 *flow_mod*/sec .

Node A generates 30 network flows, half of which are destined to node C and the rest to node D, instead, node B generates 60 flows all directed to D. We assume that the control application exploits a shortest-path algorithm to calculate the path for a given network flow, without taking into account the resource constraints on the switches. In fact, this is the current approach to path definition as implemented in many of OpenFlow controllers.

Under these conditions, the example shows that the network would not be able to support all the flows offered by the edge nodes. In particular, while node A can send all its flows to C and D (Figure 1a), ten flows sent by node B would be not forwarded by the network (Figure 1b). The problem is in the routing algorithm that forwards flows from node B to node D always on the same path, thus consuming all the available resources of the switch located at the bottom-center. The network itself has actually available resources to sustain that load, and Figure 1c offers a practical solution to take

[1]Service function chaining is the process of steering traffic through a chain of middleboxes implementing network functions, such as NAT, Firewall, etc.

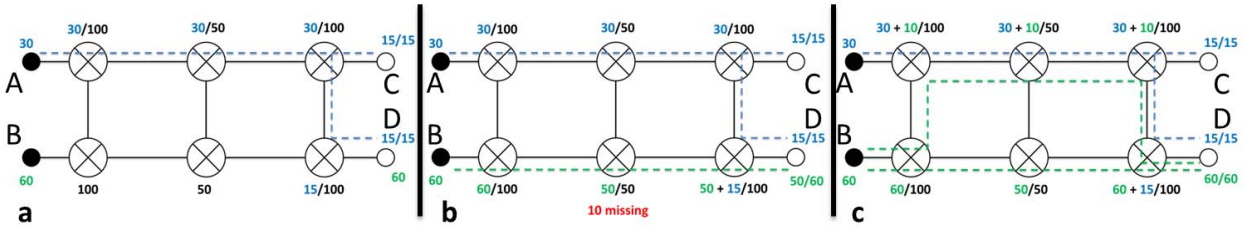

Fig. 1: a). An example of routing solutions for a network that needs to forward 90 flows from nodes A, B to nodes C, D in a given time period. b). Using shortest-path routing, 10 flows cannot be forwarded. c). A solution that shows how the network could forward those flow if adopting a different routing algorithm that takes into account switches performance limitations

advantage of them, in which the last ten flows are installed on an alternative path.

B. The capacity of the network as a whole

Knowing the capacity of a single switch can be as easy as reading its datasheet. However, knowing the theoretical limits of the network as a whole can be not as straightforward, assuming that we can exploit all the resources of our network to install flows, like in the example of Figure 1. Easy-to-pose design questions such as "what is the maximum number of flows our network can handle?" may be not as easy to answer to. Luckily, answering these questions is equivalent to solve a `maximum flow problem` over our network. The maximum flow problem is defined as follows: given a source and a destination, and a network with weighted edges, what is the maximum amount of units that can be moved from the source to the destination? The problem assumes the weights on the edges to be the maximum number of units that can traverse a given edge.

To adapt the problem formulation to our case, we first have to consider each network flow that needs to be forwarded in the network as a unit. Second, we have to add node constraints and introduce multiple sources and multiple destinations for the flows. Transformation for these cases are well known in literature. Finally, we need to assume that the flow definition is the same for all the switches, and that each flow requires the installation of the same number of flow table entries in any switch, on the selected path between the flow source and destination. To answer our network capacity problem, we just need to apply one of the efficient techniques to solve the maximum flow problem, which are already available [4].

When solving the problem for the example of Figure 1, we find that the network can support a theoretical limit of 100 *flow_mod*/sec. This limit would shrink to just 50 *flow_mod*/sec if the controller would not take into account the switches limitations[2]. As an outcome of the considerations of this section, we highlight the following takeaway.

[2]When designing a real network, there are also other limitations to consider (e. g., throughput or delay), thus, for an optimal solution we may need to solve maximum flow problems for multiple metrics, which may even impact on each other. For the sake of clarity, we ignore this additional complexity in this paper

Takeaway 1: Taking into account switches' limitations and differences can improve the network performance and/or reduce the switch resources requirements

IV. The role of network traffic

The *packet_in*s and *flow_mod*s are usually generated in response to the reception of a new network flow at some switch. Hence, to handle the switches performance limitations, the SDN control plane needs to take into account also the network traffic characteristics. To motivate our statement, we use the simulator presented in [2] to show a case in which the handling of *packet_in*s and *flow_mod*s requires particular care in order to guarantee correct network operations. Please notice that current SDN control plane implementations usually already consider the network traffic characteristics in some way, for instance to perform load balancing [21]. Anyway, in these cases the control plane is mainly considering this information for traffic engineering, e. g., to provide enough bandwidth or to respect a given maximum delay constraint. At the best of our knowledge, there are no examples in literature of control plane implementations that exploit this information to guarantee the correct operations of the network control plane itself.

A. Switch simulator

The switch simulator presented in [2] allows to estimate the number of *packet_in*s, *flow_mod*s and FTEs a switch is required to handle. It is based on the following model: a SDN controller interacts with a given switch by instrumenting it with new flow table entries, in response to network events. This means that the switch is not aware of any devices other than the controller within the network, and it does not consider other types of interaction which could happen between the control plane and the switch. Therefore, to the switch point of view, an SDN control application solely provides the flow table entries to populate and update the switch's flow table.

Starting from this simplified model of interaction between the controller and the switch, the simulator analyzes the traffic traversing the switch and keeps track of each flow. The traffic is provided either in form of an off-line trace or live packets, while the definition of the flow is configurable and is based on the header fields of, e. g., the IP and transport protocol. The

simulator further supports the specification of an idle timeout for the FTEs: if the switch does not see packets related to a given flow within this timeout period, it deletes the flow entry from its table. As output, the simulator returns the number of flows being created (i. e., an entry is added to the table), active (i. e., the table has an entry related to the flow and its timeout has not expired), or missed (i. e., the entry of a previously expired flow is added again in the table) over time. Any creation or missed generates a FT update event, which corresponds to the generation of a *packet_in* and corresponding *flow_mod* messages.

B. Application to a real-world traffic trace

The case we are considering is the design of a service for which the control plane is required to take actions based on the destination IP address, that is, the granularity of the flow is defined as all packets with the same destination IP address. To ensure the correct network operations, we need to evaluate whether or not our network would be able to support forwarding based on such flow granularity or, alternatively, whether there is a sub-portion of the network able to do it. Moreover, we may need to evaluate a suitable timeout value to allow the switch to free entries in its flow table, by trading it with the overhead in communications between the controller and the switch in terms of exchange of control messages[3].

To answer these design questions, we applied the simulator to a real traffic trace. The selected trace is a fifteen-minute capture over a 150 Mbit/s trans-pacific link between USA and Japan [11]. We assume this to be representative of the traffic traversing a switch located on a backbone segment of the network. From an initial analysis of the trace, we observe the presence of heavy host-scan activities, which leads to abruptly increase the number of flows with unique IP destination addresses, even if those flows were mainly composed of few packets. In Figure 2 we show the number of flows with less than ten packets which start at each second of our capture: a host-scan is active in the time window $[1 - 680]$ seconds and then again briefly at the end of the capture. During these periods, the number of flows started per second increases to more than ten thousands.

A host-scan is a clear anomaly, nonetheless, we need to take it into account and deal with it to guarantee the network operations. In fact, many of the advanced applications listed in Section II rely on *packet_in*s, and a host-scan activity may finally negatively affect the operations of a portion of the network. Current deployment experiences show that a switch CPU may become overloaded, affecting its ability to communicate with the controller [10]. This may eventually result in a complete outage of the switch, causing the interruption for any network flow served by such device. Some switches, such as [14], protect themselves, e. g., by throttling the number of generated *packet_in*/sec. Nevertheless, this would still result in a Denial of Service for a subset of the legitimate network flows, since the legitimate *packet_in*s would be discarded due to the high number of *packet_in*s generated because of the traffic anomaly. A more effective countermeasure to handle the host-scan would be to account for the switch limitation directly in the control plane. For example, the control plane may redirect a portion of the incoming network flows towards a sanitization middlebox, when the switch gets close to its *packet_in* generation limit.

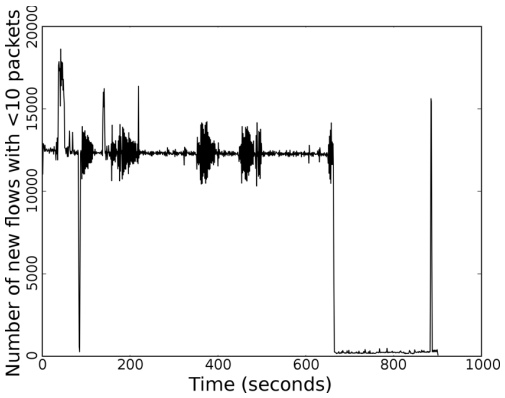

Fig. 2: Number of flows with less than ten packets started in each second of traffic trace: time series. Note the presence of two host-scan activities, one from the beginning to the second 680, and the other at the end of the trace.

Assuming that such an approach would prevent our network operations to be affected by the host-scan activity, we run our simulation on a trace where the flows responsible for the host-scan are filtered out.

We performed in total four simulations, increasing at each run the flow idle timeout by an order of magnitude, from 1s to 1000s. The reason for which we are interested in the idle timeout is twofold: on the one hand we want to ensure that control plane requirements are met, e. g., detecting when a flow terminates or ensuring no forwarding delays for long-lived flows; on the other hand we may need to trade-off the number of control messages with the number of FTEs installed in the switch's FT.

The simulator provides the status of the resources of the switch for each second of the simulation, in terms of flows being created, active or missed in the flow table. The results are shown in Figure 3, in form of time series and radar plots. To understand if a switch would be able to sustain its workload, we need to look at the maximum required values for the *packet_in*s and *flow_mod*s rates (flows being created and missed), since the actual amount of resources being used must be below the switch performance constraints at any point in time.

The first column of radar plots (leftmost one) provides, for a fixed value of the timeout, the maximum, in the output

[3]The available FT space is an additional crucial parameter when designing OpenFlow networks. We are not including it in our discussion, since it is a well known constraint to account for when designing SDN applications. The reader can find descriptions and solutions to the FT space management problem in several research papers, such as [9], [10], [17], [20]. The relationship between the flows' idle timeout, the number of FTEs and the number of control messages has been studied in [2].

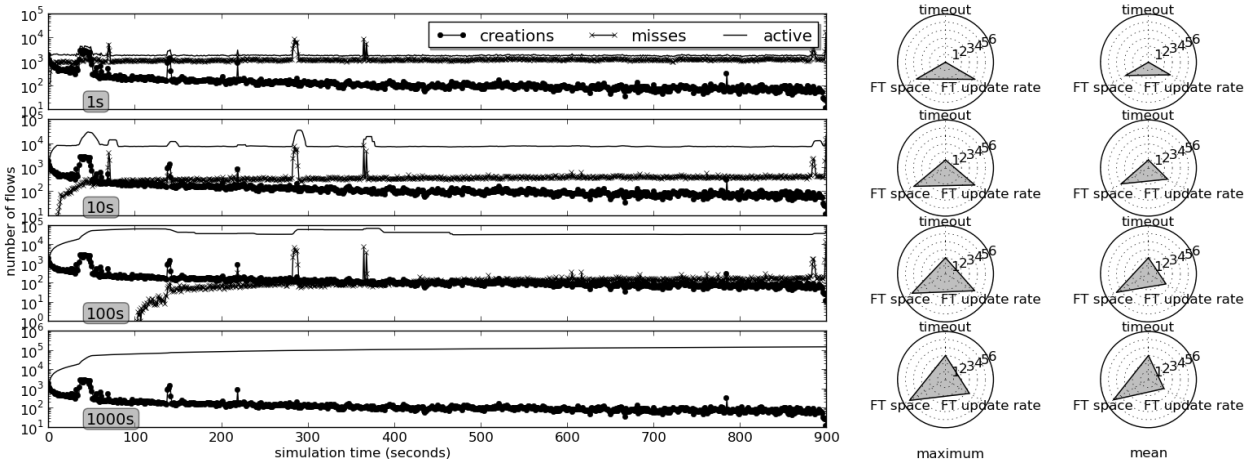

Fig. 3: On the left: Number of flows being created, active or missed over time in the flow table of the switch when considering a backbone traffic trace. Flows' idle timeout value is reported in the gray box for each of the four experiments. On the right: radar plots of the numbers of flows being created, active or missed, maximum (left column) and average (right column) values (axis in logarithmic scale).

Timeout	number of FTEs		Control messages	
$[s]$	max	mean	max	mean
1	16251	2042	18547	1431
10	37503	8982	17548	602
100	72657	42226	16720	347
1000	159588	114542	2880	177

TABLE II: Required resources in terms of FTEs and control messages when varying the flow idle timeout.

of the simulator, among the numbers of FTEs and control messages generated in a second. The rightmost column of radar plots, instead, provides the average of values over the entire simulation period. Numerical values are reported in Table II. Regardless of the timeout (and with the host-scan activities being filtered out), our switch would require to support something in the order of 10^4 control messages per second, in order to support the maximum values reported by the simulator and to be suitable for our control plane. What we observe is that in all the cases, the required control messages rates exceed the rates offered by the currently available SDN switches (Table I).

As a consequence, it appears we cannot implement a control plane with a flow definition based on the destination IP address, when the traffic characteristics are the ones of the analyzed packet trace. That is, our simulation results show that the control plane would anyway face issues due to resources shortage. However, the time series of Figure 3 show that the actual maximum value of required resources is due to few outstanding peaks, which are short in time. If we consider the average numbers of FTEs and control messages (rightmost radar plots of Figure 3), the resources requirements may reduce of at least an order of magnitude. For instance, the control messages rate drops from 17548 to 602 messages per second with an idle timeout value of ten seconds. With these values we could indeed deploy the control plane using the currently available switches, provided that the other network constraints, such as bandwidth and delay, are respected.

To handle the peaks shown in the simulation, the control plane requires to know when a given switch is close to its performance limit. Similarly to the example of Figure 1, the control plane may then perform a temporary redirection of the network traffic to leverage the available resources to other switches, ensuring that the network keeps on serving new flows, without overloading its devices. Since the peak periods are short in time, the redirection may happen for short temporary period of time, so that any sub-optimal path is taken by a redirected flow just for a short time. The considerations of this section suggest the following takeaway.

Takeaway 2: The control plane needs to monitor the switch/controller interactions and combine this information with the knowledge of the switches limitations.

V. Possible solutions

In order to tackle the issues we highlighted in this paper, the first step is learning the switches' limitations. To this purpose, the control plane designer may either trust the information provided by the vendor or run benchmarking tests on the switches to gather the performance information. In this case, the challenge is in measuring correctly a complex device, which may show heterogeneous performance depending on the workload. For instance, some FTEs may require longer time to be installed in a switch than other FTEs. To avoid long and expensive testing sessions, a possible approach is to test a device only against the expected type of workload, i.e., using the control plane logic and realistic network models to understand in advance the device limitations for the specific

deployment case. Of course, the test should aim at increasing as much as possible the workload to finally hit the switch's limitations.

Assuming that the controller knows the switch limitations and that the interactions with the switches are monitored, we envision two categories of solutions, which are explained below.

Advanced routing algorithms: a first category of solutions takes into account the switches limitations directly in the routing algorithms. That is, the switch limitations are added as additional constraint to the routing problem. The challenge of this approach is that the workload on a switch may suddenly change, in an unpredictable way, like the simulation of Figure 3 shows. To further improve the routing algorithm, it is also possible to include dynamic flow routes optimizations, when the load level on the switches changes.

Switches offloading: a different approach is using techniques that leverage additional (or unused) resources borrowed from other devices. For instance, it is possible to extend a hardware switch with a software one, in order to create a temporary buffer for flows. In fact, the higher flow table update rate usually available in a software switch can be exploited, at the cost of providing limited bandwidth for a subset of flows. Eventually, when the rate of control messages is lower, flows can be "promoted" to the hardware switch.

VI. Conclusion

In this paper we showed that taking into account the performance limitations of OpenFlow switches at handling control messages can improve the network control plane design.

We focused on a so far overlooked performance metric: how fast a switch is in handling *packet_in*s and *flow_mod*s. We introduced some applications that make use of these control messages, and the measured performance numbers for some of the currently available OpenFlow switches implementations. We then presented a theoretical example to show that taking into account these performance numbers can increase the overall network performance, e. g., in the total number of flows the network can host, or that it is in general possible to reduce the single switch resources requirements. Furthermore, we introduced a simulation-based experiment to show the importance of taking into account the traffic characteristics when dealing with control planes that extensively make use of the *packet_in* and *flow_mod* messages. In particular, the combination of this information, with the knowledge over the switches performance limitations, enables the control plane at avoiding network outages, e. g., caused by network traffic anomalies, or at better handling switches' workload peaks.

We believe that systematically addressing the issues raised in this paper, providing generally applicable methods and solutions, is a required step for the research activities in SDN and will be part of our future work.

References

[1] Bifulco, R., and Karame, G. Towards a richer set of services in software-defined networks. In *Proceedings of the NDSS Workshop on Security of Emerging Technologies (SENT)* (2014).

[2] Dusi, M., Bifulco, R., Gringoli, F., and Schneider, F. Reactive logic in software-defined networking: Measuring flow-table requirements. In *Proceedings of the 5th International Workshop on TRaffic Analysis and Characterization (TRAC)* (2014).

[3] Ferguson, A. D., Guha, A., Liang, C., Fonseca, R., and Krishnamurthi, S. Participatory networking: an api for application control of sdns. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 327–338.

[4] Goldberg, A. V., and Tarjan, R. E. A new approach to the maximum-flow problem. *J. ACM 35*, 4 (Oct. 1988), 921–940.

[5] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., and Wattenhofer, R. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 15–26.

[6] Huang, D. Y., Yocum, K., and Snoeren, A. C. High-fidelity switch models for software-defined network emulation. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (New York, NY, USA, 2013), HotSDN '13, ACM.

[7] Intel. Dpdk vswitch. https://01.org/packet-processing.

[8] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev. 43*, 4 (Aug. 2013), 3–14.

[9] Kang, N., Liu, Z., Rexford, J., and Walker, D. Optimizing the "one big switch" abstraction in software-defined networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2013), CoNEXT '13, ACM, pp. 13–24.

[10] Kobayashi, M., Seetharaman, S., Parulkar, G. M., Appenzeller, G., Little, J., van Reijendam, J., Weissmann, P., and McKeown, N. Maturing of openflow and software-defined networking through deployments. *Computer Networks 61* (2014).

[11] MAWI Working Group Traffic Archive – Trace from Apr 4th 2013. http://mawi.wide.ad.jp/mawi/samplepoint-F/2013/201304041400.html.

[12] Mehdi, S. A., Khalid, J., and Khayam, S. A. Revisiting traffic anomaly detection using software defined networking. In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection* (Berlin, Heidelberg, 2011), RAID'11, Springer-Verlag, pp. 161–180.

[13] Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., Curtis, A. R., and Banerjee, S. Devoflow: cost-effective flow management for high performance enterprise networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (New York, NY, USA, 2010), Hotnets-IX, ACM, pp. 1:1–1:6.

[14] NEC. NEC ProgrammableFlow UNIVERGE PF5240 Datasheet. http://www.necam.com/docs/?id=5ce9b8d9-e3f3-41de-a5c2-6bd7c9b37246.

[15] NoviSwitch. Noviswitch website. http://http://noviflow.com/products/noviswitch/.

[16] OpenvSwitch. Openvswitch website. http://openvswitch.org/.

[17] Qazi, Z. A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V., and Yu, M. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 27–38.

[18] Rckert, J., Bifulco, R., Rizwan-Ul-Haq, M., Kolbe, H.-J., and Hausheer, D. Flexible traffic management in broadband access networks using software defined networking. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)* (2014).

[19] Shin, S., Porras, P. A., Yegneswaran, V., Fong, M. W., Gu, G., and Tyson, M. Fresco: Modular composable security services for software-defined networks. In *NDSS* (2013), The Internet Society.

[20] Voellmy, A., Wang, J., Yang, Y. R., Ford, B., and Hudak, P. Maple: simplifying sdn programming using algorithmic policies. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 87–98.

[21] Wang, R., Butnariu, D., and Rexford, J. Openflow-based server load balancing gone wild. In *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services* (Berkeley, CA, USA, 2011), Hot-ICE'11, USENIX Association, pp. 12–12.