FLOWer – Device Benchmarking Beyond 100 Gbit/s

Paul Emmerich, Sebastian Gallenmüller, and Georg Carle
Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich
{*emmericp*|*gallenmu*|*carle*}*@net.in.tum.de*

Abstract

The growth of bandwidth in computer networks fuels the constant adoption of measurement equipment and methodologies. Networking equipment offers processing capacity in the Terabit/s range nowadays. However, test equipment in academia lags behind. We propose FLOWer, a novel and cost effective approach capable of testing such high-speed devices. FLOWer combines an inexpensive software packet generator with an OpenFlow-enabled switch to amplify the bandwidth while sustaining the flexibility of the software solution. By utilizing OpenFlow, FLOWer is able to provide the required bandwidths the software solution cannot generate on its own. We demonstrate a proof-of-concept with example measurements at bandwidths of multiple 100 Gbit/s.

1. Introduction

With data centers deploying 10 GbE as standard equipment and with the increasing availability of succeeding standards like 40 GbE and 100 GbE the demand for bandwidth in network devices is rising. At the same time, technologies like OpenFlow and network function virtualization introduce new possibilities to configure these devices. However, this increased flexibility comes with a price, as there is a tradeoff between performance and flexibility. This situation creates the need for new benchmarking and testing methodologies to enable an accurate assessment of these devices.

We propose a new way to efficiently test devices with bandwidths in the Terabit/s range with little effort. To achieve this FLOWer uses a combination of a packet generator with an OpenFlow switch. This combined system offers the flexibility of a software packet generator and at the same time the performance of a hardware solution through the OpenFlow-enabled switch. The software-generated packets are fed into an OpenFlow-enabled switch which can multiply them manifold via OpenFlow and generate high bandwidths in a simple and elegant way. OpenFlow capabilities also allow modification or the measurement of the traffic.

Our evaluation is based on modern 10 Gbit/s OpenFlow hardware. Relevant features and limitations of these devices are discussed further in Section 3. The most basic test setup uses the capabilities of FLOWer to perform self-tests on OpenFlow-enabled devices by connecting the FLOWer switch with itself. We show this setup with example measurements in Section 4. More complex test setups can be achieved by connecting the FLOWer switch to other devices. A sample setup is discussed in Section 5. All results and code used for the experiments presented here are publicly available [5], cf. Section 6.

2. Related Work

Packet generators face a trade-off between flexibility and performance. Software packet generators are typically slow and unreliable [2]. Hardware packet generators offer high precision, speed, and number of ports [21]. However, they lack the flexibility of modern software packet generators that can be configured with scripts [7]. Specialized hardware is always expensive compared to the commodity hardware required for software tools (cf. Section 3.3). The software packet generator MoonGen [7] solves this problem to some extent by using hardware features found on commodity server network interface cards (NICs) to provide high precision. However, the speed and number of ports still lags behind commercial hardware offerings as it is restricted to server hardware.

Professional hardware packet generators offer a large number of ports and even multiple 100 GbE ports [22]. The need for higher speeds is apparent in the literature. E.g., Rotsos et al. present OFLOPS, a framework to test OpenFlow switches with a NetFPGA [20]. Their original framework was limited to 100 Mbit/s on GbE ports and later extended to 20 Gbit/s [19] with the OSNT packet generator [1]. However, the devices they are testing offer speeds beyond 100 Gbit/s. FLOWer is not an alternative to existing packet generators, but an addition: it can be combined with a framework like OFLOPS to solve this discrepancy.

A technique similar to FLOWer was used by Mahadevan et al. in 2009 [11]. They wired a switch in a way that all ports were connected back to the switch itself. Broadcast traffic sent over this configuration loads the switch without requiring a high performance traffic generator. We use a similar wiring approach for self testing the FLOWer switch (cf. Section 4) However, our approach is more precise as it uses OpenFlow features to shape unicast traffic to our specific requirements and to measure throughput.

ISBN 978-3-901882-84-5, IFIP Networking 2016 © 2016 IFIP

Kuźniar et al. discuss characteristics of OpenFlow flow table implementations on different switches [10]. The results like the flow table size and costs of modification operations are important for us to select a suitable OpenFlow switch.

3. Test Hardware and Software

In the following the hard- and software is presented which we used for our proof of concept. However, FLOWer does not depend on these specific devices. It is rather a methodology and can be transferred to other OpenFlow-enabled hardware and other packet generators.

3.1. OpenFlow Switches

OpenFlow specifies programmable switches: they can be configured to match packet headers based on *flows*. These flows are also referred to as rules in this paper. The packets matching against these flows can be modified via modification actions and can then be sent out on one or more ports. Traffic can be accounted through statistics associated with flows. [13]

We use an Edge-Core Networks AS5712-54X 10 GbE switch which is based on Broadcom BCM56854 Trident II switch ASICs with 48 10 GbE ports and 6 40 GbE ports [4] for our proof of concept. This switch is a design approved by the Open Compute Project, several switches with similar designs are available on the market [12][1]. The standardized design allows for multiple choices of operating systems while keeping the hardware costs low. We selected the Pic-OS operating system as it features a mature implementation of OpenFlow 1.4 [18].

This hard- and software allows for benchmarking speeds of up to 720 Gbit/s. However, we did not have 40 GbE cables in stock, so we were restricted to 480 Gbit/s here. Even this speed and port density is beyond the capabilities of packet generators that are usually available in the academic field.

3.2. MoonGen Packet Generator

We use our packet generator MoonGen [6], as it is a highly flexible software packet generator: it crafts all packets in real-time with user-provided Lua scripts. It also features latency measurements with sub-microsecond precision by using timestamping features on commodity NICs. [7]

High-speed packet generation is not required for FLOWer, but it simplifies some test setups if the packet generator can keep up with the fastest port on the switch. We use MoonGen with two 10 GbE interfaces, which it is able to saturate on a low-end Xeon E3-1230 v2 with a dual port Intel X520-T2 NIC. MoonGen is capable of generating bandwidth of up to 180 Mpps at 120 Gbit/s on commodity hardware, so it is scalable to 100 GbE switches. Readers interested in more details about MoonGen are referred to our full evaluation. [7]

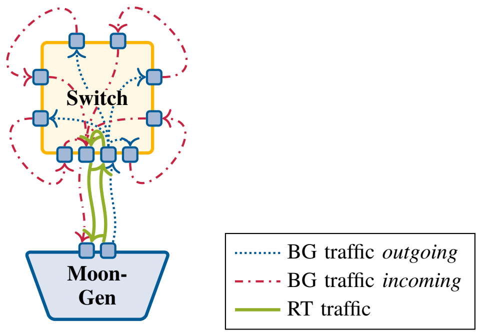

Figure 1. Self-flooding test setup

3.3. Cost Effectiveness

The total cost of the test setup (without the device under test (DuT)) to benchmark devices at 710 Gbit/s with this switch was less than €10 000. A test setups for several Tbit/s using newer 100 GbE switches can be built for less than €30 000.

These prices are based on quotes from 2015 and are expected to drop even further below prices for hardware packet generators[2] as OpenFlow switches and 40/100 GbE networks become more commonplace.

4. Self-testing SDN Devices

OpenFlow can be used for self-tests of devices. This allows testing an OpenFlow-enabled switch at maximum rate, without requiring any additional hardware beside the device under test, in this case the switch itself, and a software packet generator running on commodity hardware.

We present possible configurations and example measurements here. A switch can be wired and programmed such that it sends traffic to itself. We connected port 47 and 48 to our packet generator and port 1 with port 2, port 3 with port 4, etc.

4.1. Evaluating Quality of Service Features

Switches can define multiple queues per port with different priorities to implement quality of service (QoS). Such hardware features on switches enable modern implementations of QoS in data centers, e.g. by using IEEE 802.1Q [9] service classes mapped to queues on a switch [8]. FLOWer allows us to test hardware features like this under extreme circumstances as this example measurement demonstrates.

4.1.1. Test Setup. We generate two network flows of minimum-sized UDP packets with different destination UDP ports at our packet generator. Figure 1 merely illustrates

[1] This switch is listed as "Accton AS5712-54X" in the specification; Edge-Core Networks is a subsidiary of Accton.

[2] A search on eBay suggests that a second-hand Spirent TestCenter packet generator, without any software licenses, with a comparable number of ports and speeds costs far more than €100 000.

wiring and traffic flow not the actual setup. This sketch has a reduced number of ports and does not represent the full duplex transmission.

The first network flow, the *real-time (RT) flow*, is to be prioritized and sent with a constant rate of 1 Gbit/s. This flow is matched by a rule on switch port 48 and sent directly back to the packet generator on switch port 47 via a high-priority queue (cf. the solid line in Figure 1).

The rate of the second network flow, the *background (BG) flow*, is varied in this experiment. The switch is configured to send it out on ports 1 to 46, as depicted with dotted lines in Figure 1. From there it flows back to these ports via the external cabling (cf. the slash dotted line in Figure 1). This amplifies the traffic 46-fold. All incoming traffic from these ports is sent back to the packet generator via a low-priority queue. This tests the behavior of a prioritized network flow under increasing load of unimportant background traffic.

The packet generator then measures the latency of both network flows. Note that the packet generator receives up to 46 copies for each packet it sends in the background which is a potential challenge for timestamping. MoonGen uses sequence numbers for timestamping and defines the latency as the time until the first copy of a packet arrives back at the packet generator. Subsequent packets with the same sequence number are ignored, this is therefore the best-case latency of the background traffic.

Our repository [5] contains the script `selftest/qos.sh` which was used to install the OpenFlow flows on the DuT.

4.1.2. Test Results.

Figure 2 shows the latencies of the two network flows with the QoS queue enabled and disabled. It demonstrates that the QoS features work with hundreds of Gbit/s BG traffic.

The deviation of about 700 ns between background and real-time traffic for low rates in both tests is a result of the test setup: the background traffic flows through an additional hop during the amplification step while the real-time traffic is forwarded directly back to the packet generator (cf. Figure 1).

Another result of this test is that the RT traffic is affected by the presence of BG traffic even with QoS enabled. Inspecting the histograms of the RT traffic's latency reveals that it follows a bimodal distribution. Figure 3 shows the latency under a background load of 8 Gbit/s with two clearly visible peaks. As the switch operates in cut through mode, the left peak represents the immediate transfer of a packet. The right peak shows delayed transfer due to ongoing BG traffic transmission, resulting in this bimodal distribution.

Figure 4 on the next page shows the latency distributions for other ratios of RT to BG traffic as cumulative distribution functions (CDFs). The amount of packets that must be queued increases with the BG traffic, i.e., the ratio of the peaks in the distribution changes with the ratio of the traffic. All packets must be queued once the link is saturated, so the QoS features works best when the link is not overloaded.

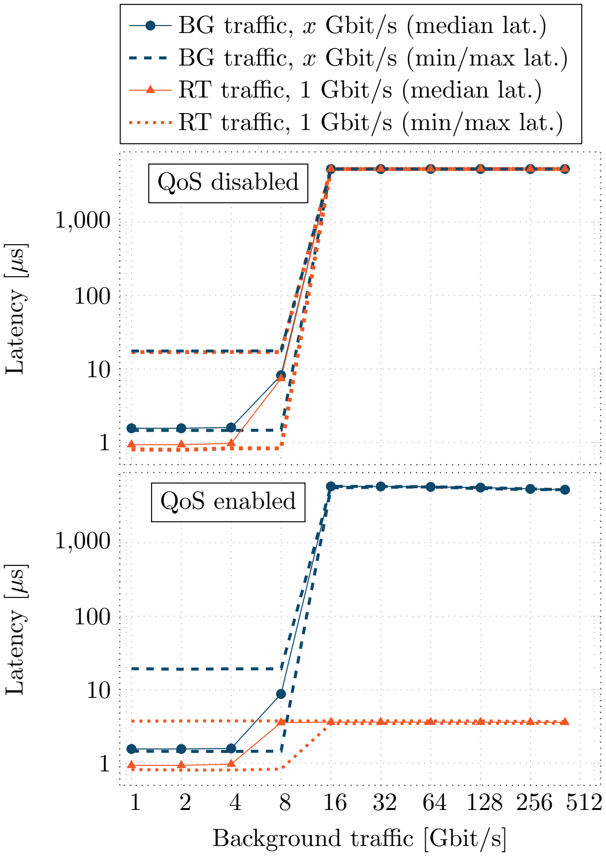

Figure 2. Forwarding latencies with and without QoS

4.2. Forwarding Performance and Latency

Another self-test scenario is forwarding packets in a loop (Figure 5). The packet generator sends on both ports to generate a total bidirectional load of 480 Gbit/s on the switch. All internal connections are realized with OpenFlow flows.

4.2.1. Forwarding Performance.

We added OpenFlow rules that match addresses from layer 1 (switch ports) to layer 4 (UDP ports) with modifications of all supported header fields from layer 2 to 4 for all packets to maximize the system load.

The switch achieved line rate in all tested configurations. Some OpenFlow switches perform a fallback to a software implementation for operations not supported in hardware at the expense of performance [20]. PicOS only accepts OpenFlow flows supported in hardware in line rate [18].

4.2.2. Forwarding Latency.

This setup can also be used to measure the forwarding latency of the switch under full bidirectional load of up to 480 Gbit/s. Load on the switch due to processing can influence the observed latency. For example, RFC 2544 requires measuring the latency of the DuT under full load [3]. The following test loads all 10 GbE ports of our switch to quantify this effect on our switch.

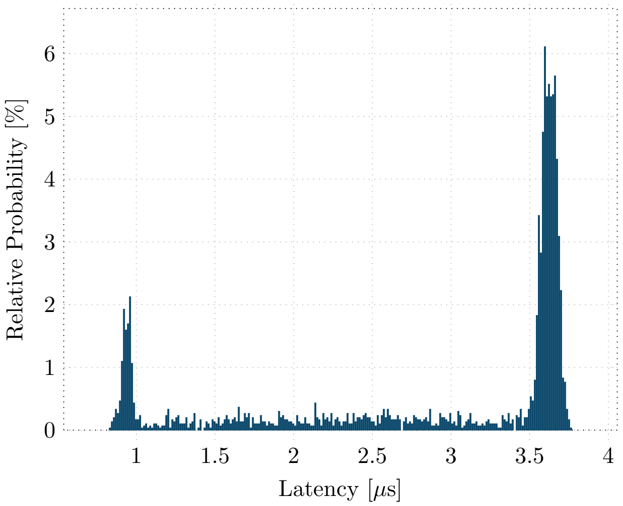

Figure 3. Latency distribution of 1 Gbit/s RT traffic with 8 Gbit/s BG traffic, QoS enabled

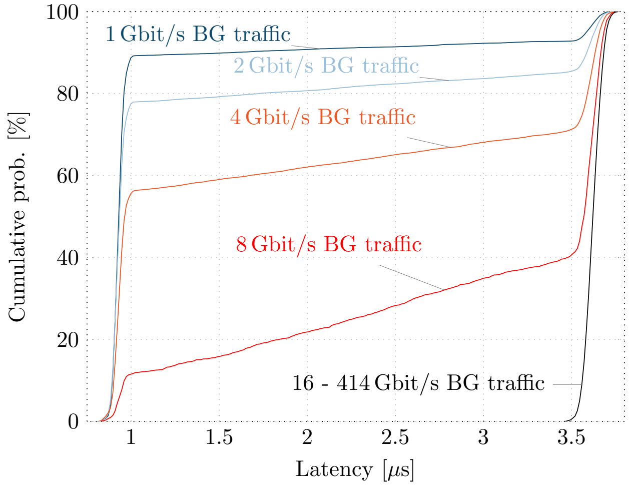

Figure 4. Latency distributions of 1 Gbit/s RT traffic with varying BG traffic, QoS enabled

The total forwarding latency l consists of the delay introduced by the connection from the packet generator to the switch l_{gen}, the forwarding latency l_{switch} of the switch, and the number of hops n:

$$l = 2 \cdot l_{gen} + n \cdot l_{switch}$$

We measured the forwarding latency through the switch with various loop lengths from $n = 0$ (sending the traffic back directly) to $n = 23$. Figure 6 shows the CDFs of different loop lengths up to $n = 15$ to improve the readability of the graph as the remaining CDFs look similar. We can calculate the following median latencies from these results: $l_{gen} = 480\,ns$ and $l_{switch} = 729\,ns$. These values include propagation delay due to varying cable lengths, we used copper cables with various lengths between 0.5 and 3 meter. This introduces an additional error of 12 ns (assuming a propagation speed of $0.7c$ [7]) in addition to the granularity of 12.8 ns of the packet generator [7].

Note that these results are crucial for FLOWer: The latency of the switch is important for further tests using the switch to amplify traffic for a separate DuT. In such a setup, the switch is part of the measurement equipment, and its accuracy therefore limits the total accuracy of the experiment.

These results show that forwarding latency does not depend on the switch ports. This indicates the high accuracy of the packet generator and that latency is independent from the used switch port. We did not test all combinations of ports, one should repeat this test with the appropriate set of ports to verify this before relying on a switch to run latency-critical experiments. There may be differences in the latency between ports on a switch due to the internal architecture of the switch.

The difference between the minimum and maximum observed forwarding latency was only 217.6 ns (cf. the steep CDFs in Figure 6, each based on 48 000 timestamped packets over 48 seconds[3]). This is important when the switch is used to amplify traffic while also measuring latency, the inaccuracy of the switch affects the measurement. OpenFlow switches with a far lower jitter exist [23] and can be used if a better precision is required.

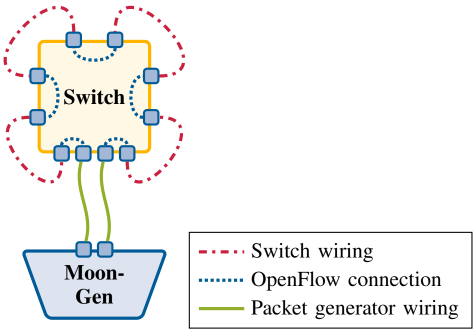

Figure 5. Loop forwarding test setup

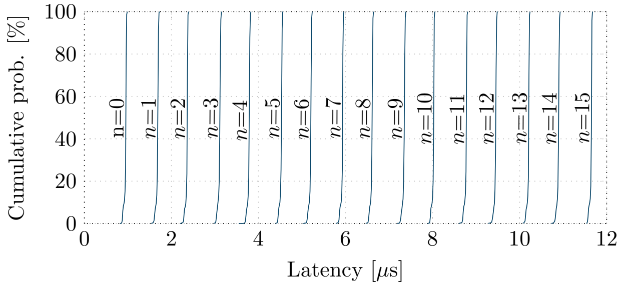

Figure 6. Latency distributions traffic forwarded through the switch n times

5. Amplifying Traffic

After evaluating the suitability of an OpenFlow Switch for our testing purposes in Section 4 we apply the FLOWer

[3] MoonGen cannot timestamp all packets, only random samples.

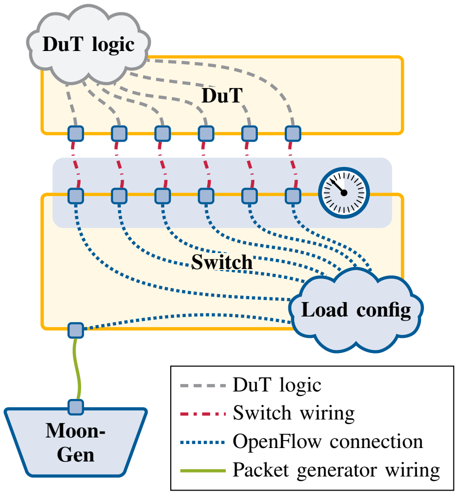

Figure 7. Testing arbitrary devices

approach to testing arbitrary networking devices. Therefore, we extend the original two-device-setup by another switch as shown in Figure 7. Both MoonGen and the DuT are connected to the switch which amplifies MoonGen's traffic and measures the throughput of the DuT. Standard tests like MoonGen's RFC 2544 [3], [24] implementation can be performed with only small modifications while increasing the bandwidth by more than an order of magnitude.

5.1. Generating Different Flows

OpenFlow flows amplify the traffic by sending incoming packets to multiple ports. Additionally, the switch can modify header fields to generate different traffic which would not be possible with traditional switches. This allows for more realistic test cases (different traffic on each port) with a multitude of network flows on different ports despite the limited output bandwidth of the packet generator. OpenFlow defines modification actions for header fields of all commonly used protocols [13] and PicOS supports most of them [18].

OpenFlow applies actions to packets, one of these actions is processing the packet via a *group* as depicted in Figure 8. Packets arrive on a port to the left, are matched via a rule and assigned to a group. Groups are meant to implement multicast and load-balancing efficiently. A group consists of multiple buckets, each defines a set of actions to be executed for packets sent to this bucket. These actions are the same that can be applied by regular rules, i.e., a group essentially clones a packet and applies different rules to each clone. Packets forwarded to the group can then be configured to be sent to either one (via hashing) or all buckets [13]. We can define a group with one bucket for each switch port and thus define modification actions that are specific to that switch port. For example, the following group table definition (in Open vSwitch syntax) shows a group that changes the IP address for each switch port, loading the DuT with completely different IP flows.

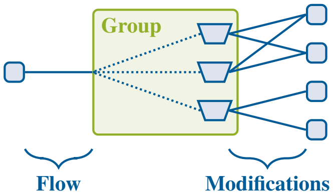

Figure 8. OpenFlow group with buckets sending to multiple output ports

```
group_id=1,type=all,
bucket=mod_nw_dst:10.0.0.1,output:1,
bucket=mod_nw_dst:10.0.0.2,output:2, ...
```

There are no restrictions for the output ports in a OpenFlow group or flow [13]. This means that a group or flow can send out a packet to a single switch port more than once by using it multiple times. This trick allows for different ratios of network flows created by the packet generator on different switch ports.

The packet generator only needs to send archetypes for various network flows which are then amplified and modified by the switch. For instance, a packet generator like MoonGen can be configured to send 5 Gbit/s TCP and 5 Gbit/s UDP traffic.

The switch can modify additional header fields to generate different traffic on different switch ports. It can even change the composition from 50:50 TCP/UDP to another ratio on some ports. For example, there can be two buckets in a group for UDP packets and none for TCP packets.

5.2. Measuring Throughput

OpenFlow counts the number of packets and bytes processed by each rule. This can be used to measure the throughput of the DuT by installing OpenFlow flows that match the traffic sent from the DuT and periodically polling their statistics. The traffic coming back from the DuT can be dropped explicitly to only count the throughput.

Statistics can be counted for multiple traffic flows by creating multiple OpenFlow flows that match on the required header fields analogous to how the switch can be used to amplify traffic flows. For example, if the switch is configured to modify the destination IP addresses for each switch port, analogous OpenFlow flows can be installed for the incoming traffic. One OpenFlow flow for each destination IP address can be installed to measure the throughput of the DuT.

Another way to measure throughput is by using OpenFlow meters. Meters are used to implement rate limiting and they also measure traffic directed to them [13]. However, a OpenFlow flow is required to send traffic to the meter and the traffic could as well be measured by the OpenFlow flow statistics. There is one scenario where this can be useful: Multiple OpenFlow flows can map to a single meter to aggregate network flows in hardware. Reading statistics can be slow, especially on switches with older CPUs [20], so this aggregation step can improve performance.

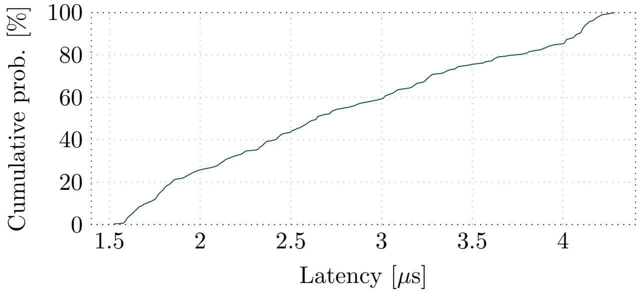

Figure 9. Latency incurred by an OpenFlow meter

Traffic can also be forwarded back to the packet generator to count it there. However, the link back is usually comparatively small here. OpenFlow meters can be used to give each network flow a fair share of the available bandwidth. This is useful to sample packets, e.g. to see if all traffic flows (e.g. wildly varying UDP ports that cannot all be installed in the hardware) get forwarded.

5.3. Measuring Latency

Latency can be measured by forwarding the packets back to the packet generator. In Section 4.2.2 we showed that the switch we used here introduces additional jitter, limiting the precision of the timestamping to ±217 ns while other switches can achieve a jitter of below 100 ns [23].

It is important to ensure no queuing happens on the amplification switch to keep the jitter at this level. The critical path is the path from the DuT back to the packet generator. Naïvely forwarding all incoming traffic to the packet generator causes queuing delay in the millisecond-range (cf. Section 4.1). The simplest solution is to filter out the timestamped packets (only a subset of the packets is timestamped) on the amplification switch and send only them back to the packet generator. A critical property of latency measurement via sampling is that the DuT cannot distinguish timestamped packets from other packets. E.g., MoonGen uses a single byte in the payload to identify timestamped packets. OpenFlow cannot match on this. However, the DuT often does not look at all header fields like the IP TTL, ECN or DSCP flags. For example, we implemented a MoonGen script[4] that marks timestamped packets by setting their TTL to 63 instead of 64. OpenFlow supports matching TTL and its exact value is usually not relevant to the DuT. Unfortunately, the switch we used does not support matching the TTL field [18], so we could not evaluate this. The actual field to hide the flag in depends on the DuT, the test case, and the capabilities of the amplification switch.

If this is not possible, e.g. due to lack of hardware support, then the only solution is using an OpenFlow meter to limit the traffic sent back to the packet generator to avoid queuing delays. Sending the traffic through a meter adds latency. Figure 9 shows the latency added by sending traffic through an OpenFlow meter which was loaded with 460 Gbit/s traffic. The difference between the minimum and maximum latency is 2.8 μs, so meters do add queuing delay, but not as much as overloading the port (cf. Section 4.1). This limits the precision of the timestamping to the μs-range.

It is of course always possible to attach the packet generator directly to some of the DuT ports, restricting the flexibility of timestamped packets. For example, MoonGen can saturate multiple 10 GbE ports, so it could load both the amplification switch and some ports of the DuT directly.

5.4. Hardware Limits

OpenFlow switches only have a limited OpenFlow flow table size. The switch we are using supports 2048 OpenFlow flows [15]. FLOWer only needs a single OpenFlow flow table entry to generate traffic and one per port to count traffic in the simplest case. More interesting scenarios need two OpenFlow flows per port and modification action: one to modify and send the packet, the other to match the modified packet and count it.

Note that the total number of different OpenFlow flows is the product of the number of modifications on the switch and the number of network flows generated by the packet generator. Modern packet generators like MoonGen allow crafting each packet in real-time through a user-controlled script, i.e., every packet can be made unique. The switch only needs one rule per output port to make every packet sent to the DuT completely unique.

The switch used here is not able to count packets on a per-flow or meter basis, only bytes. This is not a major disadvantage of this switch as the packet size is often constant or its distribution is known.

Reading statistics from the switch can also be an expensive operation on older OpenFlow switches [20]. We did not notice a significant performance degradation or CPU load when polling the statistics multiple times per second as our switch features a powerful x86 CPU.

5.5. Example: Flow Table Insertion Times

The only DuT available for this research was a second switch of the same model. A good test case for this DuT with this test setup is measuring the performance of flow table modifications under load. Using a separate DuT is a better scenario for this test than the previous self-test as the self-test requires OpenFlow flows for both the traffic generation and the tested OpenFlow flows to be on the same device. This may cause undesired interferences when attributes like insertion time or flow table size are tested. Therefore, we use the second switch as DuT here.

PicOS uses Open vSwitch to implement OpenFlow [17], so one has to understand its architecture in order to understand why this test is of particular interest.

5.5.1. Open vSwitch Architecture. Open vSwitch processes packets in two main modules: the switch daemon and

[4] Available in our repository at [5]

the datapath. The former runs on the CPU of the switch and manages all OpenFlow rules. The latter performs the actual forwarding through specialized datapath rules derived on demand from the OpenFlow rules. [14]

The datapath is implemented as a kernel module in the software-version of Open vSwitch on commodity PC hardware. PicOS replaces this datapath with a specialized version that installs the rules directly in the switching ASIC [17].

This means that an installed OpenFlow flow is not automatically and instantly available as a rule in the hardware. A packet not matching any rule on the switching ASIC is forwarded to the CPU where it is processed by the switching daemon. This daemon creates a rule for the datapath (derived from an OpenFlow flow which may contact an external controller) to match future packets of the same network flow. [14]

5.5.2. Test Setup. We connected the second switch with 32 cables to the amplification switch as shown in Figure 7. The DuT was configured with OpenFlow flows matching on all combinations of the 32 switch ports and 100 different UDP ports or addresses in separate tests, i.e., a total of 3200 network flows, that forwarded the packets back to the amplification switch where they were counted via OpenFlow flow statistics (cf. Section 5.2). MoonGen was used to generate minimum-sized UDP packets alternating between 100 different UDP ports/IP addresses. This traffic was amplified 32-fold (476 Mpps at 320 Gbit/s).

We initially forwarded samples of the packets via OpenFlow meters (cf. Section 5.3) back to MoonGen to measure the point at which a rule became active with μs-level precision. However, we noted that such a high precision was not necessary for this test as the insertion times were in the order of milliseconds per OpenFlow flow. We counted the installed OpenFlow flows with two different methods: via reading the OpenFlow statistics on the amplification switch to pinpoint the time at which the first packet arrived and via reading the hardware flow table entries directly from the switching ASIC with the `ovs-appctl pica/dump-flows` command. Both methods yielded the same results in separate experiments, eliminating potential performance impacts of the `dump-flows` command and potential delays in the OpenFlow statistics.

5.5.3. Test Results. Figure 10 shows the number of OpenFlow flows installed in the switching ASIC after sending the OpenFlow commands to add the flows. Adding all of the 3200 OpenFlow flows to the flow table in the switching daemon took only 1.5 s. The number of OpenFlow rules that can actually be realized in hardware depends on the fields used by the rule. The hardware we use features 2048 TCAM entries [15]. However, each flow table entry requires two entries if all OpenFlow features are enabled. Restricting matches to layer 1 to 3 allows using 2048 OpenFlow flows [16]. No packets were forwarded for the OpenFlow flows not present in the switching ASIC as we inserted 3200 OpenFlow flows – more than the switching ASIC supports.

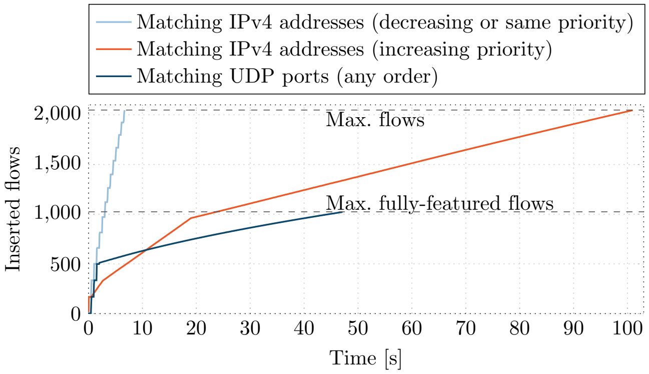

Figure 10. Flows installed in hardware over time

The insertion time may depend on the priorities of the inserted OpenFlow rules, inserting in decreasing priority is the best case and increasing the worst case [10]. We could reproduce this for rules matching on IPv4 addresses but without a significant difference between decreasing order and rules with the same priority. The insertion time was not affected by the insertion order and relatively slow for OpenFlow flows matching UDP ports. The load on the switch had no effect: We tested rates of 3.2 Gbit/s, 32 Gbit/s, and 320 Gbit/s.

The insertion time suddenly increases once the TCAM is half full at 512 or 1024 entries depending on the matched headers. Our interpretation of this result is that the hardware uses a double-buffering algorithm to speed up the required re-shuffling operations if enough space is available.

The worst-case insertion time we found was about 90 ms per OpenFlow flow for flows requiring two TCAM entries above 512 inserted flows. The best case was approximately 3 ms for IPv4 matches inserted in decreasing priority or into a half empty TCAM (6 ms for UDP port OpenFlow flows). These insertion times are significantly faster than results from older switches: Rotsos et al. found a smoothly increasing insertion time ranging from 1 s to 10 s per OpenFlow flow at 1000 flow table entries for different 1 GbE OpenFlow switches in 2012 [20]. Kuźniar et al. measured 33 to 83 ms on 1 GbE switches in 2015 [10], similar to our results.

6. Conclusions and Future Work

The results of the measurements we presented only offer incremental results over existing studies [10], [20] by using 10/40 GbE switches and vastly higher data rates. We rather introduce FLOWer as a novel approach for testing high performance network devices with minimal effort and high flexibility. This allows building high performance measurement platforms in a more cost efficient manner for either evaluating existing devices or to aid in the development of new packet processing hardware. Taking our measurement results into account, we derive the following requirements for a switch to be used with FLOWer.

- Features comparable to OpenFlow modifications including groups at line rate (some OpenFlow switches implement layer 3 or 4 matches and modifications in software, cf. [20])
- Large number of ports
- Low jitter, independent from the used port and internal packet path
- Low per-port costs, i.e., commodity hardware

Our methodology can be used with different packet generators. It is not necessarily in competition with expensive hardware packet generators. A low-end hardware packet generator can be used together with FLOWer to amplify its bandwidth while using its extensive analysis features for thousands of simultaneous network flows in hardware [21] without paying for a high-end model.

We encourage you to reproduce our measurements on your hardware. All scripts used for the experiments described in this paper and all collected data are available on GitHub [5].

In the future, we plan to use this methodology to evaluate and benchmark different devices. For this paper, only a simple SDN switch was available as DuT. More complex devices present a more interesting challenge for our methodology. In particular, we are planning to test a high-end Arbor Networks DDoS mitigation middlebox. Such DuTs are of particular interest because they are black-box software devices with high bandwidth capabilities that are beyond the reach of cheap packet generators. FLOWer puts everyone in the position to evaluate the performance of such devices without relying on expensive test equipment or vendor's promises.

Acknowledgments

This work was supported by the EUREKA-Project SASER (01BP12300A). The authors thank First Colo GmbH for providing the test hardware, Yaron Ekshtein at Pica8 for insightful discussions, and our colleagues Florian Wohlfart and Daniel Raumer for valuable contributions to this paper.

References

[1] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. McKeown, N. Feamster, B. Felderman, M. Blott, et al. OSNT: open source network tester. *Network, IEEE*, 28(5):6–12, 2014.

[2] A. Botta, A. Dainotti, and A. Pescapé. Do you trust your software-based traffic generator? *IEEE Communications Magazine*, 48(9):158–165, 2010.

[3] S. Bradner and J. McQuaid. Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational), March 1999.

[4] Edge-Core Networks. AS5712-54X Datasheet. http://www.edge-core.com/temp/ec_download/1555/AS5712-54X%20ONIE%20%20DS%20add%20Cumulus%20Ready%20Logo%20R02%20.pdf. Visited: 2015-11-23.

[5] P. Emmerich. FLOWer Scripts. https://github.com/emmericp/FLOWer-scripts.

[6] P. Emmerich. MoonGen. https://github.com/emmericp/MoonGen.

[7] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *Internet Measurement Conference 2015 (IMC'15)*, Tokyo, Japan, Oct. 2015.

[8] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can jump them! In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 1–14, Oakland, CA, May 2015. USENIX Association.

[9] IEEE. Virtual LANs. *IEEE 802.1Q-2011*, 2011.

[10] M. Kuźniar, P. Perešíni, and D. Kostić. What you need to know about sdn flow tables. In *Passive and Active Measurement*, pages 347–359. Springer, 2015.

[11] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A Power Benchmarking Framework for Network Devices. In *IFIP Networking*, pages 795–808. Springer, 2009.

[12] Open Compute Project. Networking Specs and Design. http://www.opencompute.org/wiki/Networking/SpecsAndDesigns. Visited: 2015-11-23.

[13] Open Networking Foundation. OpenFlow Switch Specification Version 1.4.0, October 2013.

[14] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, May 2015. USENIX Association.

[15] Pica8. Flow Scalability per Broadcom Chipset. http://www.pica8.com/document/v2.6/html/hardware-guides/#6914398. Visited: 2015-11-23.

[16] Pica8. Optimizing TCAM Usage. http://www.pica8.com/document/v2.6/html/ovs-configuration-guide/#6914964. Visited: 2015-11-23.

[17] Pica8. PicOS Overview. http://www.pica8.com/documents/pica8-whitepaper-picos-overview.pdf. Visited: 2015-11-23.

[18] Pica8. Supported OpenFlow Protocol 1.4 Features. http://www.pica8.com/document/v2.6/html/ovs-configuration-guide/. Visited: 2015-11-23.

[19] C. Rotsos, G. Antichi, M. Bruyere, P. Owezarski, and A. Moore. OFLOPS-Turbo: Testing the Next-Generation OpenFlow Switch. In *European Workshop on Software Defined Networks (EWSDN)*, 2014.

[20] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. OFLOPS: An Open Framework for OpenFlow Switch Evaluation. In *Passive and Active Measurement*, pages 85–95. Springer, 2012.

[21] Spirent. HyperMetrics FX 2/4/8 Port 10 Gigabit Ethernet Test Module. http://www.spirent.com/~/media/Datasheets/Broadband/PAB/SpirentTestCenter/STC_HyperMetrics_fX_2-4-8-Port_10G_Ethernet_Test_Module.pdf.

[22] Spirent. HyperMetrics FX 40/100 Gigabit Test Modules. http://www.spirent.com/~/media/Datasheets/Broadband/PAB/SpirentTestCenter/STC_HyperMetrics\%20_fX_40-100G_Module_datasheet.pdf.

[23] Tolly Enterprises, LLC. Mellanox SwitchX-2 (SX1036) vs. Broadcom StrataXGS Trident II (Arista DCS-7050QX) – Performance Evaluation. http://www.mellanox.com/related-docs/products/Tolly-215111-Mellanox-SwitchX-2_Performance.pdf, 2015.

[24] P. Werneck. RFC 2544 Tests for MoonGen. https://github.com/emmericp/MoonGen/pull/98.