On the Accuracy of Leveraging SDN for Passive Network Measurements

Michael Jarschel, Thomas Zinner, Thomas Höhn, Phuoc Tran-Gia
University of Würzburg, Institute of Computer Science, Würzburg, Germany.
Email: {michael.jarschel,zinner,thomas.hoehn,trangia}@informatik.uni-wuerzburg.de

***Abstract*—Network Measurement has emerged as one promising field of application for Software Defined Networking. The reason for this is that the logically centralized control plane of an SDN network inherently has to aggregate network state information in order to function. This circumstance can be leveraged for network measurements at the SDN controller without the need for additional equipment or active – and possibly disruptive – measurements in the network itself. However, the accuracy and potential resource overhead of this approach has not been discussed. In this paper we compare an SDN-based solution to actual traffic measurements in order to determine its accuracy and resource demand by performing tests in an OpenFlow testbed.**

***Keywords*—SDN, Network Measurement, OpenFlow**

I. Introduction

In today's networks the monitoring of Quality of Service (QoS) parameters like bandwidth, packet loss, and delay is essential to ensure the smooth operation of multimedia applications as well as the control of service-level agreements and fault detection. This is often done using a measurement setup that actively sends traffic through the network. However, this approach requires expensive special purpose measurement equipment. Furthermore, such a measurement often can only be performed in off-peak hours as an active measurement probe could disrupt critical production traffic. Thus, only limited statements are possible for the QoS experienced during business hours from inside the network. While today the application itself can actively monitor a subset of its own QoS parameters, it can not directly influence the network.

The introduction of SDN gives the network the ability to passively perform network-wide QoS measurements relying on the actual production traffic. This is possible using other techniques, but the SDN approach essentially turns network measurements into a primitive function of the network itself, eliminating the need for additional devices and allowing for a more representative view of the network state by increasing deployment flexibility. This in turn can then be used as direct input for SDN network control.

In this paper, we investigate how accurately a purely SDN-based approach can measure network parameters compared to a full reference packet trace. Furthermore, we highlight the method's potential cost and implementation difficulties. We do this by performing measurements using the SDN-based approach in an OpenFlow [1] testbed, while simultaneously mirroring and capturing the measurement and control traffic. We aim to answer the question if and in which case a network operator can benefit from a purely SDN-based measurement setup.

The remainder of this paper is structured as follows. In Section II we provide background information to SDN and discuss related work in the field of measurements and monitoring. We then introduce the measurement architecture in Section III. The testbed setup and test scenarios are presented in Section IV. Section V discusses the results with regard to the different measurement parameters. Finally, we derive our conclusions in Section VI.

II. Background and Related Work

We give a short introduction to Software Defined Networking in this section as having a basic idea about how SDN works is key to understanding the measurement approach we discuss.

A. Introduction to Software Defined Networking

The main principle behind SDN is to provide an open interface to the forwarding hardware in networks as described in [2]. The goal is to be able to directly influence the forwarding process of a network element using a freely programmable control software, thus no longer relying on proprietary management and control systems. This has the prospect of leading to a faster pace of innovation in the network as well as more competition on the market reducing the costs for network operators. To achieve this, a pure SDN switch does not have any conventional control-plane functionality but fully relies on the external controller entity to make forwarding decisions. Current SDN realizations often rely on the OpenFlow protocol standardized by the Open Networking Foundation [3] for the communication between data- and control-plane. Each OpenFlow-enabled device contains so called "flow tables", which hold a set of forwarding or "flow rules". Contrary to conventional switches, a flow rule does not match a single address but a specific flow represented by a match consisting of e.g. physical, network, and transport layer header fields. The flow rule itself consists out of this match for a specific flow, a corresponding action, and statistical counters. Once a network packet arrives at an OpenFlow switch, it is buffered and its header is extracted and matched against the flow rules in one or more flow tables. If a match is found, the action or actions defined by the rule is executed, e.g. forwarding or dropping the packet, and the flow statistics are updated. If no match can be found in any table, the packet can either be dropped or the header information is encapsulated into an OpenFlow "packet-in" message and sent to the controller. The controller determines the appropriate action for the packet and sends it back to the requesting network element for execution. Additionally, the controller can derive a flow rule, which specifies an action for all packets of the same flow. When

the new flow rule arrives at the network element, it is entered into a flow table and matched against arriving packets. For SDN-based measurements specifically, two OpenFlow features are important. First, the ability of the OpenFlow controller to query the per-flow packet and byte counters via an OpenFlow "stats request" message and second, the possibility to define the controller as the target of an output action at the switch, which enables the redirection of packets to it. For further details we refer to the OpenFlow specification [1].

B. Previous Works on (SDN-based) Measurements

In [4] Zseby evaluates sampling methods for passive QoS-measurements in conventional networks and highlights the challenges when implementing such an approach. It is these challenges that an SDN-based approach appears suitable to levy.

In recent years there have been several works that investigate ways on how to leverage SDN and specifically OpenFlow for network measurements and monitoring. Tootoonchian et al. [5] propose an OpenFlow-based approach for traffic matrix estimation by intelligently querying flowtable counters. We evaluate the accuracy of these kind of queries for bandwidth measurements.

In [6] Jose et al. investigate the possibility to measure large traffic aggregates in commodity switches, which leads to Yu et al. [7] introducing the measurement architecture OpenSketch, which, similarly to the OpenFlow concept, separates the measurement data plane from the control plane. While this is an interesting approach, we focus on the SDN control plan itself represented by an OpenFlow controller.

Yu et al. [8] introduce the FlowSense concept, an OpenFlow-based approach to network measurements with minimal measurement costs. We extend this approach by also taking latency measurements into account, which we discuss in the following Section III.

III. Measurement Architecture

In addition to bandwidth measurements, our extended SDN measurement architecture based on FlowSense [8] also takes one-way delay measurements into account. Figure 1 illustrates the concept on the example of a connection in an intermediary SDN network from switch A to switch B. All switches in the network are connected to the SDN controller via a control channel. This connection is used for switch control on the one hand and on the other for the polling of statistics information from the switches' flow tables.

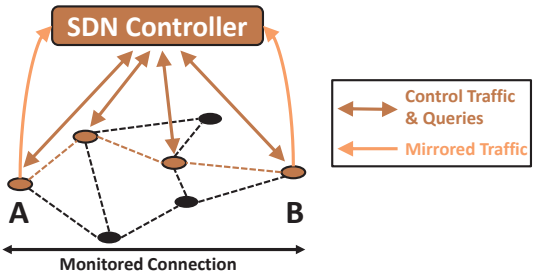


Fig. 1. Measurement Architecture

The flow tables contain packet and byte counters for each entry. By retrieving their current values the controller can calculate the current bandwidth consumed by a traffic flow matching an individual rule. In this case, that is the bandwidth consumption of our flow on all highlighted links between switches from A to B. No additional components are required in the network to measure bandwidth consumption and the information about it can be directly used to influence the controller's policy for the network or to optimize the placement of the controller(s) within the network [9]. However, this method requires frequent queries to the individual devices in order to be accurate for a desired interval.

For the purpose of delay measurements, the methodology is different. Suppose the goal is to measure the delay on the connection from A to B in our example network. In traditional networking, a mirror port would have to be configured on the ingress and egress device, that would then send all the traffic to a measurement station, which would than have to filter the data for the desired flow information and calculate the delay between the packets received from A and B. With the SDN approach, the controller can simply insert a temporary flow rule into switches A and B to send all or a sample of the packets to the controller parallel to forwarding them through the network. This way the controller again becomes the measurement device and can directly react on the network information without additional equipment.

This approach has several difficulties. We do not know how accurately the software controller can measure and calculate bandwidth and delay without the support of special purpose hardware. Furthermore, the load on the control channel may be considerable using this approach and it is unknown how, if at all, this impacts network control. It is the goal of this paper to answer some of the questions regarding accuracy.

As an alternative to the above described methods, a hybrid approach between conventional an purely SDN-based measurements is possible. In this approach, the flexibility of SDN is used to selectively mirror network flows to a dedicated measurement device instead of the controller. However, this method requires an additional special purpose device and the accuracy is determined by the implementation that device itself. Therefore, the focus of this paper lies on the purely SDN-based approach.

IV. Testbed Setup

We use an OpenFlow-based testbed to evaluate the accuracy and overhead of the purely SDN-based measurement approach described in Section III. The testbed is shown in Figure 2. It realizes a simplified version of the scenario in Figure 1. Iperf [10] is used to send a 1 Mbps UDP flow from the traffic generator to the traffic sink representing the production traffic that should be measured. The flow passes through two Pica8 Pronto 3290 switches, which represent the ingress and egress nodes of our intermediary network. Bandwidth and delay variations experienced in the network are emulated using NetEm [11] on a Linux PC. We use Floodlight [12] with a custom measurement module as OpenFlow controller running on a Dell Poweredge 860 server. The SDN-based measurements are performed using this controller. For the delay measurements, the OpenFlow switches send the traffic to

the controller as well as to its destination using two OpenFlow output actions. Bandwidth is measured by regularly sending OpenFlow statistics requests to the switches. The reference measurements are performed in parallel on a separate HP Proliant DL320 server using either an Endace DAG 7.5G2 capture card or a conventional network card in conjunction with TCPdump. The traffic is mirrored to this server using two Netoptics wire taps.

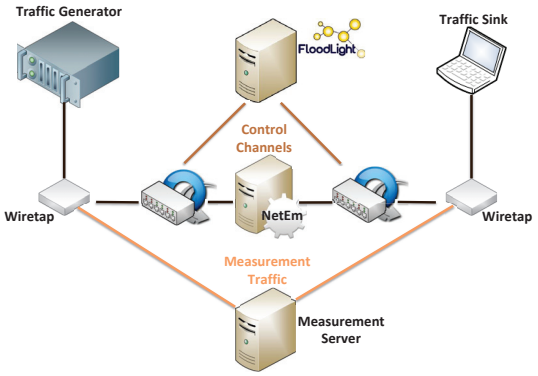


Fig. 2. Testbed Setup

Technical Considerations and Limitations:

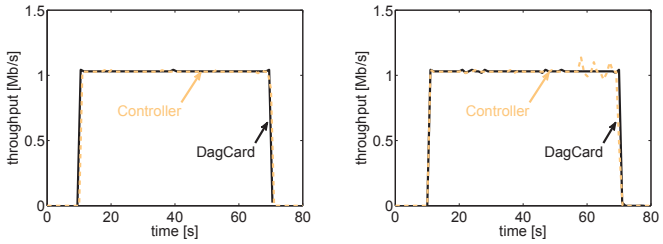
As the traffic for the SDN as well as the reference measurement is mirrored at different locations, a discrepancy in the measured delay is expected. This discrepancy is caused by the processing delay of the two OpenFlow switches. Furthermore, the measurement probe can not exceed bandwidths of much more than 1 Mbps as the OpenFlow implementation on both OpenFlow switches handles the implementation of an OpenFlow send-to-controller action in software, i.e. on the slow path, which is limited by the relatively slow switch CPU. Future OpenFlow switch implementations will likely not be constrained by this issue. We discuss the impact of this in the following section.

Another influence factor on the accuracy of the measurements is the latency on the links of the control channels between the OpenFlow switches and the controller (cf. Figure 2). Before a packet arriving at the controller from one of the two switches can be timestamped, it has already experienced additional delay from processing at the switch and the transmission via the control channel. This is also true in the conventional measurement approach we have chosen as reference and can only be avoided, if the packets are timestamped at the switches and the internal clocks of the switches are precisely synchronized. In our case, therefore an additional requirement has to be met. The term $|(\Delta t_{process_1} + \Delta t_{propagate_1}) - (\Delta t_{process_2} + \Delta t_{propagate_2})|$, where $\Delta t_{process_x}$ reflects the processing time in the switch and $\Delta t_{propagate_x}$ is the propagation delay on the control channel, has to be smaller than the desired measurement accuracy. We meet this requirement in our testbed by using identical OpenFlow switches and control channel cabling. In a real world deployment, it is also likely that identical hardware would be used and the impact of latency could be kept small by using a distributed controller and placing an instance close to the measurement point.

For the bandwidth measurements, the frequency of updates is limited to one second intervals as the OpenFlow switches only update the statistics counters in their flow tables once every second.

In an SDN deployment, it is likely that the controller would be run on a virtual machine inside the cloud. Therefore, we have performed our tests with the controller running either on the aforementioned server or in a virtual machine hosted on an identical server using the free version of the VMware ESXi 5.1 hypervisor. Particularly delay measurements require precise timekeeping in order to be accurate. As the SDN controller is run in software relying on the system's hardware clock, this can not always be guaranteed. We expect this to be even more of an issue in a virtual environment with not only different processes but virtual machines competing for processing time.

The scalability of the SDN measurement approach is limited by two factors. These are the processing capacity of the controller and the control channel bandwidth. As the controller would likely be run in a cloud environment for large setups, the processing capacity can be scaled up dynamically to the required level. However, when a distributed controller approach is used to achieve this, where different switches are connected to different controller instances, those instances require a clock synchronization. The control channel bandwidth required for the measurements can be reduced using sampling techniques. However, the number of flows that can be monitored simultaneously will still have an upper limit.



(a) Non-Virtual Controller (b) Virtual Controller

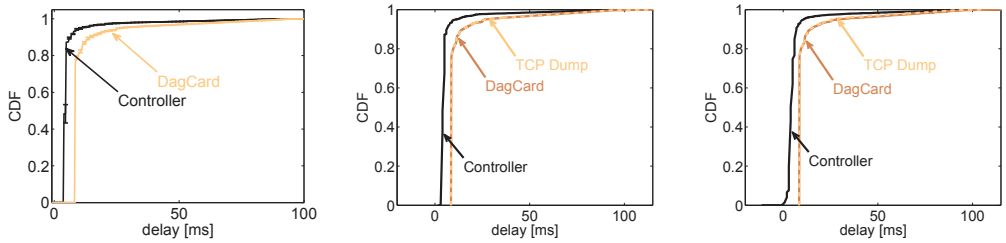
Fig. 3. Used Bandwidth

V. Measurement Results

In this section, we discuss the results of our measurements. All measurement runs were repeated at least five times in order to ensure consistency.

A. Measuring Bandwidth

As a base test for our setup we chose a bandwidth measurement using the already mentioned statistics requests. This test is very similar to those performed with FlowSense. Therefore, we use it to verify our methodology and setup. Figure 3 shows the measured bandwidth consumed by the measurement flow over the duration of a 60-second test run for both the virtual and non-virtual controller. For comparison, all packets were captured using the DAG card in the measurement server. As can be seen in the figure, in both cases the measured throughput reaches the configured 1 Mbps of the



(a) With Confidence Intervals, Non-Virtual Controller (b) Non-Virtual Controller (c) Virtual Controller

Fig. 4. Latency Cumulative Distribution Functions (No Artificial Delay)

measurement probe and subsides after the traffic generator has stopped sending packets. The SDN measurements behave nearly identical to the capture trace of the test run. While this is true for the mean of all runs performed, we can observe some inaccuracies in the particular run shown in Figure 3b. At around the 55 second mark, the bandwidth measurement at the virtual controller varies for several kbps around the reference value. There are two possible explanations for this behavior. It could be caused by time drift of the virtual machine clock, which is only synchronized with the server's hardware clock at specific intervals. Therefore, the controller could no longer reliably schedule its statistics requests and the query interval varies slightly leading to inaccurate bandwidth calculations. The second explanation is that either the query or response packets for the statistics in question were delayed by either the virtual switch in the hypervisor or by the management plane of the OpenFlow switch. However, since we do not have accurate timestamps for the control channel messages, it is not possible for us to determine which is the case. Still, the variation appears minor and thus the approach appears to be usable at least for bandwidth measurements at this frequency. A more granular resolution would require the switches to support more frequent counter updates and would involve significantly more queries to the switches' minimal control plane. This would require a much more powerful switch CPU to handle these more frequent requests and sufficient bandwidth on the control channel.

B. Measuring Latency

In this section, we discuss the delay measurements. Figure 4 shows the cumulative distribution functions (CDFs) of the delay measurements performed without the introduction of any artificial delay between the two measurement points. Figure 4a shows the results for the non-virtual controller and the DAG card with 95 percent confidence intervals for five individual runs. We observe a measured delay of 4-5 ms for about 86 percent of packets with the controller, whereas we see almost double that delay on the capture trace for 78 percent of the packets.

As can be seen, the confidence intervals are small, indicating a good estimation of the delay probabilities. The exception is the ratio of packets with 4-5 ms delay as measured by the non-virtual controller. Here, the confidence interval is in a range of about 10 percent difference for the runs. However, this shows that our results are statistically stable. Therefore, we only use one exemplary test run for each test in the remainder of the figures in order to enhance readability.

As described in Section IV, our OpenFlow switches handle packets with a send-to-controller action in software, which explains this considerable delay imposed on the packets. The discrepancy between controller and capture trace is caused by the difference in measurement points in our testbed. While the delay, measured using the controller only consists of the sending process in the first switch and the receiving process in the second, the capture trace sees the delay imposed by the sending and receiving processes of both switches, which doubles the imposed delay. For confirmation of this circumstance, we perform measurements with the hybrid approach described in Section III, using the OpenFlow switches to mirror traffic to the DAG card. Measuring at the same locations in the network, we can determine the impact of the send-to-controller action by running the tests with the action enabled and without. Figure 5 shows the cumulative distribution functions of the results. As expected, a clear discrepancy between the two curves is visible. Whereas almost all packets from the measurement without the send-to-controller action experience a delay of less than 1 ms, the packets with the send-to-controller action enabled show a delay of 6-7 ms and above. The additional latency of the value measured here to that in 4 can be explained by the fact, that the switch now has to perform three actions in software instead of two, i.e., forward packet, forward to controller, and forward packet to the DAG card.

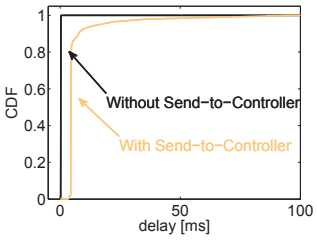


Fig. 5. Latency Cumulative Distribution Functions (With and without Send-to-Controller)

Figure 4b shows the results of a single test run for the non-virtual controller. In addition to the capture trace of the DAG card, a trace using just TCPdump is also shown. The behavior is very similar to the one observed in Figure 4a. We can see that with this amount of switching delay, the high time resolution of the DAG capture card does not present a significant advantage over a conventional TCPdump trace.

As expected, the results for DAG card and TCPdump do not differ greatly from these in the test using the virtual controller as shown in Figure 4c. However, we observe that the increase in the CDF graph for the delay measured with the virtual controller is not as steep as with the non-virtual controller. There is a visible gradient. About 84 percent of the packets are measured with a switching delay of 1-5 ms, which is a significantly greater value range than the 4-5 ms measured for the non-virtual controller. Furthermore, if we look at the lower end of the CDF plot, we see that the CDF does not start at 0 ms as can be seen in more detail in Figure 6. A small but visible percentage of packets appears to have experienced a negative delay. Naturally, this cannot have happened in reality. Therefore, a measurement error has to be responsible. This result seems to confirm our theory from Section V-A that inaccurate time keeping in the virtual machine causes irregularities in the results. If processing issues at either the virtual hypervisor switch or the OpenFlow switches were responsible, the delay would have to remain positive at all times even if it varied greatly.

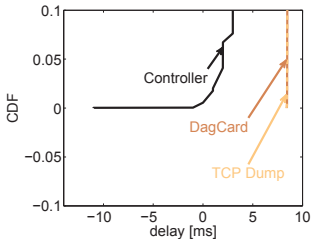
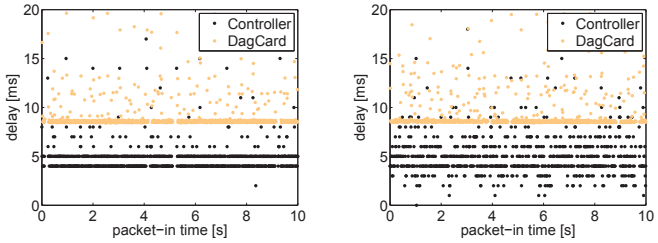


Fig. 6. Latency CDF Zoomed (Virtual Controller)

In order to understand these results better, we compare two 10 seconds long time series of samples measured with both controllers and the DAG card. Figure 7a shows the results for the non-virtual controller. Three distinct lines of often occurring delays are clearly visible. One at about 8 ms for the DAG Card and two at 4 and respective 5 ms for the non-virtual controller. Additionally, there is a similar number of outliers for both measurement methods. For the virtual controller the samples shown in Figure 7b show a different behavior. While the samples of the DAG card remain similar at around 8 ms with outliers, the samples for the virtual controller show more frequent occurring delays at 1,2,3, and 6 ms. However, while the virtual controller appears to regularly measure a broader range of delays, the coefficient of variation for both controllers is next to identical at around 1.5, whereas the DAG card has a coefficient of variation of 1. The same is true for the Mean and the Median at 6.3 ms and 4-5 ms respectively. This tells us that even though the virtual controller appears to be more volatile and does have occasional time keeping issues, statistically those shortcomings carry no weight.

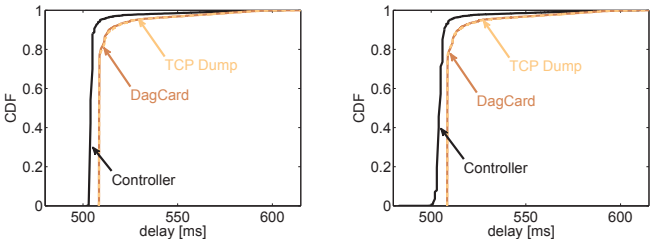
Based on these results, it appears feasible to obtain mean delay values using the purely SDN-based approach. However, for a production deployment, the switches again would have to improve their performance for applying the send-to-controller action to a packet. As our results have shown, this could be circumvented by giving the controller a secondary network interface and using a conventional output action. However, this can only serve as a temporary fix, if at all.



(a) Non-Virtual Controller (b) Virtual Controller

Fig. 7. Latency Samples

Up to this point, we have not introduced any artificial delay into our measurements. Therefore, we set our network emulator between the two switches to impose a delay of 500 ms on the measurement probe to verify the accuracy of the measurements at a higher latency level. The CDFs of the measured delays are displayed in Figure 8. The results mirror those shown in Figure 4 for both the non-virtual and virtual controllers, albeit with an offset of the configured 500 ms delay. Therefore, we can conclude that the introduction of artificial delay has had no impact on the accuracy of the results as well as on the discrepancy between controller- and server-based measurements.



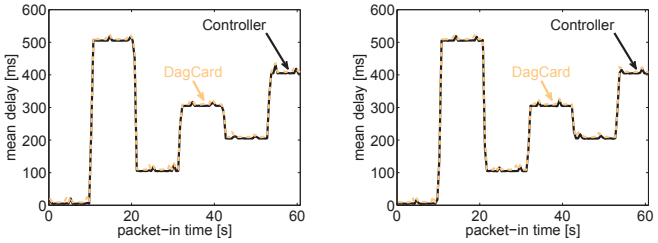
(a) Non-Virtual Controller (b) Virtual Controller

Fig. 8. Latency Cumulative Distribution Functions (500 ms Delay)

Now that we have established that the SDN-based measurement approach can indeed deliver delay measurement results statistically comparable to those of special purpose equipment in a stable environment, we take a look at what happens, when the delay in the network changes. Therefore, we program a series of delay changes into our network emulator and observe whether the changes in delay are noticed on time by the controller and whether accuracy is impacted. Figure 9 shows a 60 seconds time series of a test run with five changes to different values of delay. We observe that both controllers are able to closely mirror the delay present in the captured packet trace.

C. Sampling

As it is likely not very prudent to redirect all measurement traffic to the controller across the SDN control channel, which is also needed for network operation, the option of only redirecting a sample of packets to the controller seems



(a) Non-Virtual Controller (b) Virtual Controller

Fig. 9. Mean Latency (Variable Delays)

viable. Therefore, we take a look at how closely the full reference delay value can be estimated using only a sample. Figure 10 shows the relative error for the mean delay in relation to the sampling ratio. The relative error has been obtained by repeatedly selecting samples randomly from the full reference measurement. It can be seen that in order to limit the relative error to five percent, the DAG card only requires about five percent of the packets, whereas virtual and non-virtual controller alike require about 10 percent of the sample due to the higher volatility of the measurement results. This means, that for a 95 percent accurate result using the SDN-based approach a sample twice the size of the e.g. the hybrid approach is required, which is a considerable overhead. This again emphasizes the importance of control channel bandwidth for the SDN-based approach.

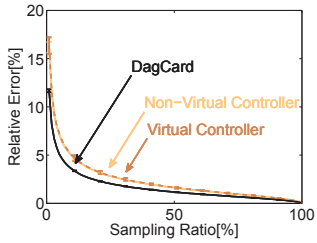


Fig. 10. Relative Error through Sampling

VI. Conclusion

In this paper we have compared the accuracy of purely SDN-based network measurements to that of a full reference packet capture trace using special purpose hardware. The results show that, while the accuracy for an individual delay sample falls behind that of the reference measurement, the mean delay and especially mean bandwidth are on par with the capture card. Therefore, if the mean value of a QoS parameter is sufficient input for the operation of a particular network, SDN-based measurements appear to be a viable and cost-efficient alternative. The inherent flexibility of SDN to mirror and redirect traffic on a per-flow basis greatly simplifies the measurement setup and in combination with virtualization can enable rapid deployments and tear downs on-demand. However, this is only possible if the SDN-enabled hardware is further developed to support this kind of function, sufficient control channel bandwidth is available, and the latency imposed by switch processing and control channel transmission is sufficiently even for both measurement points. Our results show that otherwise the measurements would become inaccurate and, more significantly, could disrupt the operation of the network. Our experience with the used OpenFlow 1.0 switches suggests that the improvement of SDN hardware is still a challenge. However, our results also indicate that the described SDN hybrid approach may serve as a working stepping stone towards pure SDN measurements.

References

[1] "OpenFlow Switch Specification, Version 1.3.2," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.2.pdf, April 2013, last accessed on 2013.06.19.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[3] "Open Networking Foundation," https://www.opennetworking.org/, last accessed on 2013.06.19.

[4] T. Zseby, "Deployment of sampling methods for sla validation with non-intrusive measurements," in *Proceedings of Passive and Active Measurement Workshop (PAM 2002), Fort Collins, CO, USA*, 2002, pp. 25–26.

[5] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: traffic matrix estimator for openflow networks," in *Passive and Active Measurement*. Springer, 2010, pp. 201–210.

[6] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proc. of the USENIX HotICE workshop*, 2011.

[7] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," Tech. rep., USC, www-bcf. usc. edu/˜ minlanyu/tech/opensketch. pdf, Tech. Rep., 2012.

[8] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *Passive and Active Measurement*. Springer, 2013, pp. 31–41.

[9] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks," in *25th International Teletraffic Congress (ITC)*, Shanghai, China, Sep. 2013.

[10] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The tcp/udp bandwidth measurement tool," *htt p://dast. nlanr. net/Projects*, 2005.

[11] S. Hemminger, "Network emulation with netem," in *Linux Conf Au*. Citeseer, 2005, pp. 18–23.

[12] "Floodlight," http://www.projectfloodlight.org/floodlight/, last accessed on 2013.06.13.