Available Bandwidth Measurement in Software Defined Networks

Péter Megyesi*, Alessio Botta†‡, Giuseppe Aceto†‡, Antonio Pescapè†‡, Sándor Molnár*
*High Speed Networks Lab., Budapest University of Technology and Economics
†University of Napoli Federico II
‡NM2 SRL (Italy)
*{megyesi, molnar}@tmit.bme.hu, †{a.botta, giuseppe.aceto, pescape}@unina.it

ABSTRACT

Software Defined Networking (SDN) is an emerging paradigm that is expected to revolutionize computer networks. With the decoupling of data and control plane and the introduction of open communication interfaces between layers, SDN enables programmability over the entire network, promising rapid innovation in this area. The SDN concept was already proven to work successfully in cloud and data center environments thus the proper monitoring of such networks is already in the focus of the research community. Methods for measuring Quality of Service (QoS) parameters such as bandwidth utilization, packet loss, and delay have been recently introduced in literature, but they lack a solution for tackling down the question of *available bandwidth*. In this paper, we attempt to fill this gap and introduce a novel mechanism for measuring available bandwidth in SDN networks. We take advantage of the SDN architecture and build an application over the Network Operating System (NOS). Our application can track the topology of the network and the bandwidth utilization over the network links, and thus it is able to calculate the available bandwidth between any two points in the network. We validate our method using the popular Mininet network emulation environment and the widely used NOS called Floodlight. We present results providing insights into the measurement accuracy and showing its relationship with the delay in the control network and the polling frequency.

CCS Concepts

•**Networks** → **Programming interfaces; Network performance evaluation; Network measurement;** *Network architectures; Network algorithms;*

SAC 2016,April 04 - 08, 2016, Pisa, Italy

ACM 978-1-4503-3739-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851727

Keywords

Software Defined Networks, available bandwidth, OpenFlow, Network Operating System, Floodlight, mininet

1. INTRODUCTION

Today computer networks are everywhere. In our everyday life we are almost always connected to the Internet and, in most cases, we are also connected in our working hours since many business-critical applications also need network connection. The different demands of heterogeneous networks has led to a situation where nowadays IP networks are very complex to both build and manage. Current network architectures are rigid thus it is especially hard to add new features to them. Software Defined Networking (SDN) offers a solution for this problem mainly through the following features: i) data and control planes are decoupled; ii) control logic is moved out of the network devices (SDN switches) to an external Network Operating System (also called the SDN controller); iii) external applications can program the network using the abstraction mechanisms provided by the SDN controller. The SDN concept has quickly gained significant focus by the research community after the introduction of OpenFlow in 2008 [1]. In the last few years there have been several proposals for monitoring Quality of Service (QoS) parameters in SDN networks. They mostly tackle the problems of e.g. bandwidth utilization [2–6], packet loss ratio [6], packet delay [6,7], and route tracing [8]. On the other hand, we could not find any paper in literature that deals with the problem of available bandwidth (ABW) measurement in SDN. However, ABW measurement can have significant importance for both service provider and application perspectives. Service providers use this parameter for network management and traffic engineering purposes. Furthermore, nowadays, video streaming generates the largest portion of Internet traffic, where ABW measurement techniques play a significant role in adapting to the current network load. In general, knowledge about the available bandwidth over the network would benefit many users and operators of network applications and infrastructures.

The main contributions of this paper are to present a method for available bandwidth measurement in Software Defined Networks, and to discuss the related trade-offs, specific of this architecture. Taking advantage of the features that SDN offers, we present a solution for ABW measurement in three different scenarios: (i) when paths in the network are

out of our control and we want to know the route between two points and the associated available bandwidth, (ii) when paths are under our control and we want to find the highest available bandwidth between two arbitrary points in the network and the associated route, (iii) the same scenario as the previous one in multipath environment and we want to find the best possible multipath solution. We also validate our method on a test environment using the Mininet network emulation tool [9] and the Floodlight SDN controller [10]. We estimate and discuss the impact of delays in the control network over the accuracy and relate it with polling frequency, providing to the best of our knowledge the first evaluation of this important aspect. The reminder of this paper is structured as follows. Sec. 2 presents a short background on Software Defined Networks and the related work. In Sec. 3 we present our method for measuring available bandwidth in SDN. Sec. 4 describes the test configuration we used to validate our method (Sec. 4.1) and the results of tests (Sec. 4.2). Finally, Sec. 5 ends the paper with concluding remarks.

2. BACKGROUND AND RELATED WORK

Software Defined Networking gained significant focus after the introduction of OpenFlow [1]. However, its main concepts root in earlier works in the fields of active networks, control and data plane separation, and network virtualization. In this paper we follow the definition of SDN as presented in [17], which are based on the following four elements: (i) Control and data planes are separated from each other. Network devices no longer have control functionalities, they become simple forwarding elements. (ii) Forwarding rules are made based on a set of fields in the packet headers. This also guarantees unified behavior of networking elements such as switches, routes or firewalls. (iii) Control plane is moved to an external entity called the Network Operating System (NOS) or SDN controller. NOS is a software platform that runs on commodity hardware and can communicate the forwarding rules to the switches via open standards. (iv) Third party applications can program the network over the NOS. The controller must also provide the necessary abstractions and interfaces for serving these applications. The SDN controller can communicate with the switch over the control network via the southbound API, where the most used standard is OpenFlow, and there are also other proposals, e.g. OVSDB [14], POF [15] or ROFL [16]. For NOS platform there are many available open software such as NOX [11], POX [11], Floodlight [10].Moreover, there are ongoing industrial consortia projects for controller platforms specialized for data centers, for e.g. OpenDayLight [12] or OpenStack [13]. SDN applications can program the network using the northbound API of the NOS. However, these APIs are specific to the controller thus most of the currently available SND applications are only able to operate over one NOS platform. We refer to [17] for a comprehensive taxonomy of different elements in Software Defined Networking.

In the recent years, there has been several proposals for monitoring Software Defined Networks. FlowSense authors [2] propose to use only the mandatory OpenFlow messages to monitor the bandwidth utilization over the network. Although this approach offers bandwidth monitoring with zero extra load to the network, it has been proven to work inaccurately under dynamic traffic conditions [4]. Other papers propose to use the *FlowStatsReq* message in OpenFlow to poll the interface and flow counters in the switches for bandwidth measurement [4–6]. Furthermore, PayLess [4] and MonSamp [5] offer adaptive sampling algorithms that can adapt for the current network load. However, their approaches are conflicting since PayLess suggests to increase the sampling rate when the traffic load is high (for increasing the accuracy), whereas MonSamp suggests to decrease the sampling rate under high load (so the higher the network load the lower monitoring load should be generated).

OpenNetMon [6] offers a solution for loss and delay monitoring as well. For loss measurement, it polls the flow counters on the ingress and egress switches for a given flow and calculates the difference. For delay measurement, it uses the SDN controller to inject probe packets into the network along a given path and than loop them back to the controller. The tool is able to calculate the delay for the given path using the round trip time between ingress and egress switches. Phemius and Bouet [7] use the same approach for delay measurement, but observe a constant difference between the measured and reference time values. They also present a method to calculate this value and calibrate the delay measurement accordingly. We found that the current literature lacks a solution for measuring available bandwidth in SDN. Available bandwidth is an important dynamic characteristic of a network path, being equivalent to the amount of traffic that can be added to the path without affecting the other flows that traverse part of it, and independently from their bandwidth-sharing properties. Such definition tells it apart from other bandwidth-related metrics such as *bulk transfer capacity* and from the *maximum achievable throughput* [18]. In traditional networks, available bandwidth estimation techniques are typically classified into active and passive. Passive techniques use multiple measurement points in the network to monitor bandwidth utilization, packet loss ratio, and packet delay. These techniques are very complex to deploy in traditional networks thus they are rarely used in practice. Active techniques send probe packets into the network and analyze how network traversal affected their spacing/arrival to infer network status. On the basis of the hypotheses on the analyzed path and on the type of probing procedure adopted, active ABW estimation techniques in the literature can be referred to two models: *probe gap* and *packet rate*. Probe gap tools such as Spruce or Traceband use packet pairs as probes, and require knowledge of link capacity. Probe rate tools use multiple series of packets, injected at different rates, aimed at causing a temporary congestion. Examples of probe rate tools include PathLoad and PathChirp. The performance of most of active ABW estimation tools currently available is scenario-dependent and require non-trivial calibration [19, 20]. In our approach we use a passive technique taking advantage of the NOS in the architecture of SDN. We use the northbound API to discover the topology of the network and to monitor the bandwidth utilization of the links. With this information we calculate the available bandwidth for any path in the network in any given time.

3. MEASURING AVAILABLE BANDWIDTH IN SDN NETWORKS

In our available bandwidth application we take advantage of the network abstractions provided by the NOS. Using the northbound API of the SDN controller we query all the switches inside the network and the links between them. Firstly, our application uses this information to build up a network topology graph $G(V, E)$, where the node set V corresponds to the switches and the edge set E corresponds to the links. In such model, $P_{A\to B}$ is the set of all available paths from A to B, e_i is the i^{th} link in the network topology graph, c_i is the capacity of e_i, b_i its current bandwidth load and a_i the related available capacity ($a_i = c_i - b_i$). Furthermore, we assume that the *capacity* c_i of every link is known in the network. This is a viable assumption since the type of every link is known by the NOS, and if any further policy limits the bandwidth on a given link the network operator should have information about it. The application is also able to measure the *current load* b_i of every link. For this we use an approach similar to the one previously presented in [4–6]: we periodically poll the counters in the SDN switches using the *FlowStatsReq* OpenFlow message. We calculate *current load* $b_i(t)$ at time t as

$$b_i(t) = \frac{n_i(t) - n_i(t-T)}{T}. \quad (1)$$

where $n_i(t)$ is the counter value and T is the polling period. After this step, we calculate the *available capacity* a_i on every link in the network. Based on the a_i values we then calculate the available bandwidth on a given path P through the following equation

$$ABW_P = \min_{e_i \in P} a_i. \quad (2)$$

Differently from ABW estimation in traditional networks, the method we adopt is not affected by estimation error related to link utilization. The source of error in our case is mainly related with time: the instant the flow counters are read on the switch is unknown, as there is no timestamp field in the OpenFlow spec. In our current implementation, we approximate such time instant with the arrival time of the *FlowStatsReq* message at the controller. This approximation is directly reflected on the duration of the averaging interval, and thus on the *current load* (see Eq.1). We evaluate its impact empirically in the following section.

Our method is also able to distinguish between three different scenarios and calculate the ABW according to them. They are the following: **ABW on fixed paths.** In this scenario the routing policies are fixed. Thus for a given flow, first we have to find out its route on the network, and then calculate the available bandwidth using (2).

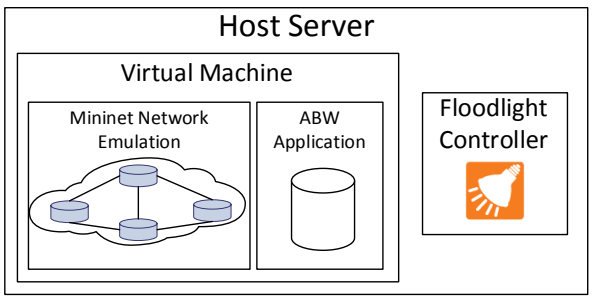


Figure 1: The assembled test configuration.

Table 1: Configuration hardware and software.

Host CPU	Intel Xeon E5-2640 v2 @ 2.00GHz
Host Memory	32 GB
Host OS	Ubuntu 14.04, Linux kernel 3.13.0-24
Virtualization	VirtualBox 4.3.20
Guest OS[1]	Ubuntu 14.04 64-bit
VM configuration	4 CPU cores, 2 GB memory
Mininet version	2.2.0
Floodlight version	1.0

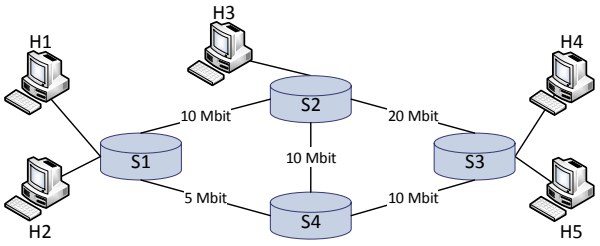


Figure 2: The test topology in Mininet.

Our method uses the northbound API of the NOS for this task, e.g. Floodlight's REST API provides an interface for reporting the route of a flow in the network (for any given header on a given entry point) according to the policies set up in the controller.

Best available path. In this case we have to find the path P between two points in the network where the available bandwidth is the largest. This can be calculated through the following equation:

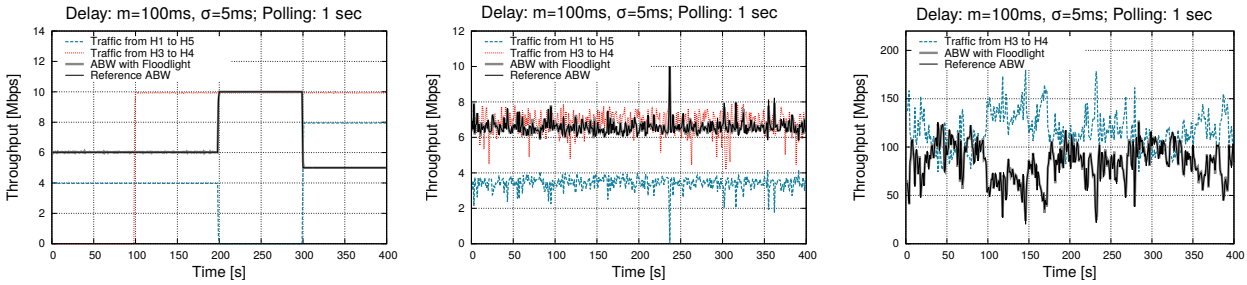
$$ABW_{A\to B} = \max_{P \in P_{A\to B}} \min_{e_i \in P} a_i. \quad (3)$$

For solving this equation we use a modified Dijkstra algorithm where the metric of a path is not measured by the sum of the edge capacities (distances) but by (2). This algorithm also gives the best possible path for the best AWB solution in $O(|E|+|V|\log|V|)$ (like a standard shortest-path Dijkstra algorithm would do).

Multipath scenario. In this case we can use multiple paths between two points in the network. We consider this as an important scenario since the SDN architecture can easily enable solutions for multipath routing, for e.g. using MPTCP in the transport layer [21]. In this case we face off a classical max-flow problem over the network topology graph $G(V, E)$ which can be solved through the Ford-Fulkerson Algorithm in $O(|E|f)$ complexity (where f is the maximum flow in the graph).

4. EXPERIMENTAL RESULTS

4.1 Test Configuration



(a) CBR traffic measurement. (b) VBR traffic measurement. (c) Real traffic measurement.

Figure 3: Measuring bandwidth on the link between S2 and S3 and the available bandwidth over the Mininet test network.

Fig. 1 presents the schematics of our testbed. We use Mininet for network emulation and Floodlight [10] as SDN controller. Mininet is running inside a virtual machine[1] on the host server using VirtualBox. We chose to run Floodlight directly in the host server since we often faced load issues when we run it inside the virtual machine. For further reference, we collected the used hardware and software versions in Tab. 1.

Fig. 2 sketches the network topology we created in Mininet for our tests. S1, S2, and S3 create a classical Y topology which is frequently used as a testbed for testing ABW applications [19]. The idea is to set up the link between S1 and S2 to serve as the bottleneck link (lowest capacity on the path) and then use H3 to generate cross traffic on link between S2 and S3. If this cross traffic is high enough, the bottleneck link and the tight link will become different, which is an important test case for the calculations of current ABW tools [19]. To realize such scenario, we use *TrafficControl* to maximize the bandwidth capacities of the links and also, (in some scenarios) to add variable delay between the switches and the Floodlight controller.

The default route policy in Floodlight will not allow to send any traffic through S4. This enables us to use the feature in our application which can predict the best possible alternative route even if such route is not the default one. If the volume of cross traffic from H3 to H4 is larger than 10 Mbps, the alternative route through S4 would provide path with larger available bandwidth.

Furthermore, we use D-ITG [22–24] for traffic generation, which was proven to work much reliably than other traffic generation platforms [25]. Using D-ITG, we were able to define the following three traffic scenarios:

CBR Traffic. In this scenario we generate three flows with constant bit rate with the following timing. At the beginning of the measurement, H1 starts to send 4 Mbps of UDP traffic to H5 for 100 seconds, than the host sleeps for 100 s (generating no traffic) and restarts sending with 8 Mbps rate. Parallel to this, H3 starts to send 10 Mbps of UDP traffic to H4 for 100 s after the start of the measurement until the end. Fig. 3a presents the traffic from H1 to H5 and from H3 to H4, and also the available bandwidth between H1 and H5. Although the bandwidth measured by the Floodlight controller is varying due to the variable delay introduced between the switches and the NOS, in some cases the measured ABW is constant. This can happen when the alternative route through S4 provides a better ABW solution: e.g. in the last 100 s of the measurement, the bandwidth between S1 and S2 is 8 Mbps thus the best path is S1→S4→S3 with 5 Mbps available bandwidth.

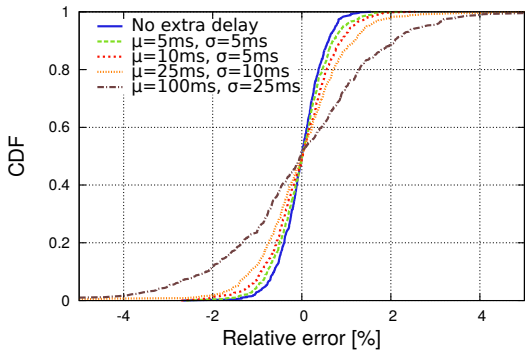
VBR Traffic is generated by D-ITG using Pareto distribution for the inter-departure times of packets. We generated two flows, one from H1 to H5 and the other one from H3 to H4. We tested different values of shape and scale parameters and report the most interesting cases in the following. Fig. 3b plots the traffic from H1 to H5 and from H3 to H4, and the available bandwidth between H1 to H5. In this case we used $\lambda = 1.75$ as shape parameter for both flows, whereas for scale parameter we used $X_{H1} = 1ms$ and $X_{H3} = 0.5ms$ for H1 and H3, respectively. As shown, the error of the ABW measurement is larger than in case of CBR traffic using frequent polling rates. On the other hand, since the inter-departure times of packets are identically and independently distributed on larger time scales, the traffic becomes smoother making the error rate similar to CBR results.

Real Traffic. We are able to collect real traffic measurements from the campus network of the Budapest University of Technology using a Cisco 6500 Layer-3 switch that aggregates the traffic of two buildings and links them to the core layer of the network. The 10-minute-long trace we used for this purpose was recorded in Nov 2013 and contains about 12 million packets, 10 GB total data, 3000 individual users and 170k flows. We extracted the inter-packet times and packet sizes from the trace and set up D-ITG to send the same traffic from H3 to H4. Since this traffic rate is much higher than the one we used for the previous cases, we also increased the capacity of the links tenfold. Fig. 3c presents generated traffic and the ABW measurement in this scenario. In this case the throughput also varies in larger time scales, thus we expect to measure higher ABW error rates using larger polling frequency.

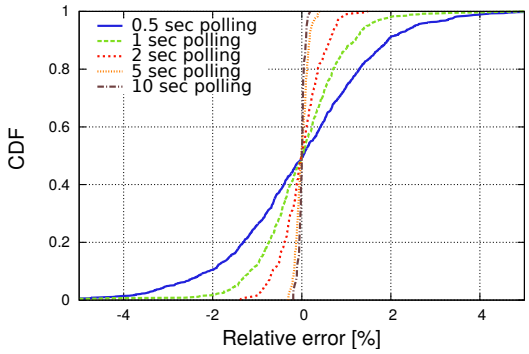
D-ITG uses logging mechanism which we used for further reference, tracking of the inter-departure times of packets. Moreover, hosts are configured with the CPU isolation method introduced in Mininet HiFi [26], thus they can not interfere with the traffic generation process of each other.

4.2 Results

[1]Virtual machine image was downloaded from Mininet website: https://github.com/mininet/mininet/wiki/Mininet-VM-Images



(a) Error values for 1 ms polling using different delay.



(b) Error values for 25ms delay using different polling periods.

Figure 4: The CDF of the relative errors of the available bandwidth measurements using the Floodlight controller compared to reference values.

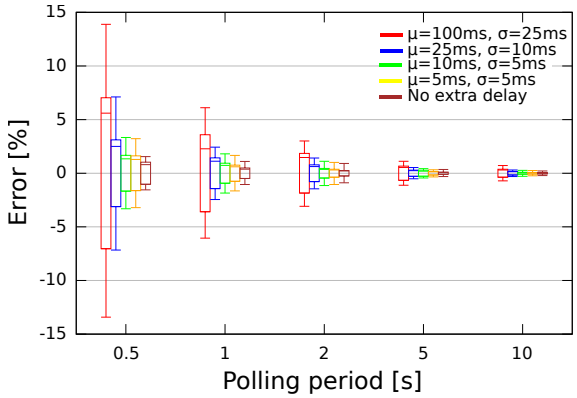


Figure 5: Box plot of relative error values for measurements using real traffic. We plotted the min and max values, the 75% percentile and the average of the absolute values.

As a first step we conducted measurement on the Mininet environment using a wide range of polling rates and without adding any delay, thus emulating an ideal case. The first row of Table 2 reports the mean and the standard deviation of the measured error values: they represent errors intrinsic to the measurement setup, due to the fact that it is not possible to know the timestamp of the counters provided by the switches in the current OpenFlow spec. However, we notice that the average error rate with 0.5 sec polling is under 1%, and the error is further decreasing with the increase of the polling period. The reason for this decrease is that with larger polling interval we average on a larger time scale while the difference in the timestamp approximation remains the same, thus having a smaller relative effect. If we consider the minute time scale (not shown in table), where typically the current ABW tools operate [20], the average error rate is less than 10^{-5} which can be considered as very accurate.

Hereafter, we introduce artificial delay between the switches and the Floodlight controller to investigate its effect over the error rate. Furthermore, the presented cases correspond to the real measurements since the results were quantitatively very similar in the other two scenarios. Fig. 4 shows the CDF of the error for different delay values using frequent polling rates. In Fig. 4a we fixed the polling period to 1 s and used different delay values between the switches and the Floodlight controller. In details, we added delay values

Table 2: Error rate of ABW measurements using real traffic replayed by D-ITG.

		Polling period in seconds									
		0.5		1		2		5		10	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Switches → NOS delay	no delay	0.72%	0.94%	0.39%	0.51%	0.19%	0.23%	0.09%	0.11%	0.05%	0.06%
	$\mu = 5ms$ $\sigma = 5ms$	1.29%	1.62%	0.6%	0.75%	0.3%	0.37%	0.13%	0.16%	0.06%	0.07%
	$\mu = 10ms$ $\sigma = 5ms$	1.34%	1.67%	0.73%	0.93%	0.36%	0.45%	0.19%	0.28%	0.07%	0.09%
	$\mu = 25ms$ $\sigma = 10ms$	2.5%	3.11%	1.12%	1.43%	0.62%	0.78%	0.24%	0.29%	0.14%	0.16%
	$\mu = 100ms$ $\sigma = 25ms$	5.6%	7.04%	2.28%	3.59%	1.47%	1.85%	0.54%	0.66%	0.3%	0.37%

following a normal distribution, with mean values of 5 ms, 10 ms, 25 ms and 100 ms and standard deviation of 5 ms, 5 ms, 10 ms and 25 ms, respectively. Here, one can investigate the effect of monitoring packet delay on the error rate of the ABW measurement. Interestingly, even with very large delay values (e.g. mean value of 100 ms) the error is usually smaller than 4% using 1 sec polling period. Using smaller delay setups the resulting curves are close to each other.

In Fig. 4b we show only one delay value (25 ms mean with 5 ms standard deviation) and used the following polling rates to calculate the available bandwidth: 0.5 s, 1 s, 2 s, 5 sec and 10 sec. The results confirm that increasing the polling period the measurement error decreases since the uncertainty on the time remains the same while the measurement interval increases, thus the relative effect of delay will be smaller. As a consequence, increasing the polling period we can achieve more precise ABW values in case we do not need very frequent results. This leads to the conclusion that the proper value for polling rate and maximum delay acceptable is a function of the application that is in need of the ABW estimation. Some applications (e.g. for streaming server selection) may require infrequent but accurate estimations. Others (e.g. for routing) may require frequent estimation, tolerating a lower accuracy.

Tab. 2 summarizes the mean and the standard deviation of the measurement result in the most interesting cases. Interestingly, when the standard deviation of the delay is set to 5 ms the resulting error values for 5 ms and 10 ms mean delay are really close to each other (2nd and 3rd rows in Tab 2). This suggest that the variance of the delay has larger impact on the measurement error. We will investigate this phenomenon in our future work. On the other hand, increasing the polling period decreases the measurement error. For further reference we also created a box plot based on these result in Fig. 5. Here we plotted the min and max error values along with the quartiles, and we also marked the average error rate of the absolute values. Fig. 5 and Tab. 2 clearly show the trade-off constrains between the error rate, the polling frequency, and the monitoring packet delay. Applications working with SDN networks and in need of ABW estimations can be properly devised looking at these results.

5. CONCLUSION

In this paper we presented for the first time in literature an approach to measure end-to-end available bandwidth (ABW) in Software Defined Networks (SDN). Our proposal uses the Network Operating System (NOS) to generate a graph representation of the network topology and then uses OpenFlow messages to track the bandwidth utilization of every link in the network. Based on this information, it is able to calculate the ABW on every path in the network. We implemented and validated our application by means of a testbed using Mininet for network emulation and Floodlight for SDN controller. We reported results obtained in different network configurations, with different traffic types (CBR, VBR, and real traffic), and with different random delays between the switches and the controller (to reproduce different possible SDN deployments). Our results show that our approach provides accurate results if compared with the ground truth. These results also constitute a reference for ABW applications willing to operate in SDN environments, which require a proper trade-off between accuracy, polling rate, and network delay constrains. In our ongoing work we are investigating the possible impact of the specific implementation of the NOS on the accuracy and timeliness of estimations, as well as applications of our technique to hybrid scenarios mixing SDN and traditional networks.

Acknowledgment

This work is partially funded by the MIUR in the context of Art. 11 DM 593/2000 for NM2 SRL.

6. REFERENCES

[1] N. McKeown et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communnication Review*, 38(2):69–74, Mar. 2008.

[2] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. Madhyastha. Flowsense: Monitoring network utilization with zero measurement cost. In *Passive and Active Measurement*, volume 7799 of *Lecture Notes in Computer Science*, pages 31–41. 2013.

[3] M. Jarschel, T. Zinner, T. Hohn, and P. Tran-Gia. On the accuracy of leveraging SDN for passive network measurements. In *Australasian Telecommunication Networks and Applications Conference 2013 (ATNAC '13)*, pages 41–46, Nov 2013.

[4] S. Chowdhury, M. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium*, pages 1–9, May 2014.

[5] D. Raumer, L. Schwaighofer, and G. Carle. Monsamp: A distributed sdn application for qos monitoring. In *Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sept. 2014.

[6] N. van Adrichem, C. Doerr, and F. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–8, May 2014.

[7] K. Phemius and M. Bouet. "Monitoring latency with openflow". In *9th International Conference on Network and Service Management (CNSM)*, pages 122–125, 2013.

[8] K. Agarwal, E. Rozner, C. Dixon, and J. Carter. Sdn traceroute: Tracing sdn forwarding without changing network behavior. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, pages 145–150, 2014.

[9] B. Lantz et al. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 19:1–19:6, 2010.

[10] Floodlight. retrieved: Sept., 2015.

[11] NOX and POX SDN Controllers. retrieved: Sept., 2015.

[12] OpenDayLight Project. retrieved: Sept., 2015.

[13] OpenStack Project. retrieved: Sept., 2015.

[14] B. Pfaff and B. Davie. The Open vSwitch Database Management Protocol, RFC7047. https://tools.ietf.org/html/rfc7047.

[15] H. Song. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In *Proc. of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 127–132, 2013.

[16] M. Sune, V. Alvarez, T. Jungel, U. Toseef, and K. Pentikousis. An openflow implementation for network processors. In *Third European Workshop on Software Defined Networks (EWSDN)*, pages 123–124, Sept 2014.

[17] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[18] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *Network, IEEE*, 17(6):27–35, 2003.

[19] A. Botta, A. Davy, B. Meskill, and G. Aceto. Active techniques for available bandwidth estimation: Comparison and application. In *Data Traffic Monitoring and Analysis*, volume 7754 of *Lecture Notes in Computer Science*, pages 28–43. 2013.

[20] G. Aceto, A. Botta, A. Pescapè, and M. D'Arienzo. Unified architecture for network measurement: The case of available bandwidth. *J. Network and Computer Applications*, 35(5):1402–1414, 2012.

[21] B. Sonkoly et al. SDN based testbeds for evaluating and promoting multipath tcp. In *Proc. IEEE International Conference on Communications (ICC 2014)*, pages 3044–3050, June 2014.

[22] S. Avallone, A. Pescapè, and G. Ventre. Distributed internet traffic generator (d-itg): analysis and experimentation over heterogeneous networks. In *International Conference on Network Protocols, Atlanta, Georgia*, 2003.

[23] D. Emma, A. Pescapè, and G. Ventre. Analysis and experimentation of an open distributed platform for synthetic traffic generation. In *Proc. FTDCS 2004.*, pages 277–283, May 2004.

[24] A. Botta, A. Dainotti, and A. Pescapè. A Tool for the Generation of Realistic Network Workload for Emerging Networking Scenarios. *Computer Networks*, 56(15):3531 – 3547, 2012.

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

[26] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proc. of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, pages 253–264, 2012.