Bandwidth Estimation Between End Hosts in SDN Technical Report

Péter Megyesi, Zoltán Mózar, Sándor Molnár
High Speed Networks Lab., Dept. of Telecomm. and Media Informatics,
Budapest University of Technology and Economics, Budapest, Hungary
E-mail: {megyesi, moczar, molnar}@tmit.bme.hu

***Abstract*—Software Defined Networking is an emerging paradigm that is expected to revolutionize the building of computer networks. With the decoupling of data and control plane and the introduction of open communication interfaces between the networking layers, SDN enables programmability over the entire network promising rapid innovation in this area. The SDN concept was already proven to work well 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 QoS parameters such as bandwidth utilization, packet loss and delay were already introduced in the literature, but they lack of a solution for tackling down the question of available bandwidth. It 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 an SDN controller that can track the topology of the network and the bandwidth utilization over the links, and an by this able to calculate the available bandwidth between two points in the network. We also validate our method using the popular Mininet network emulation environment.**

I. Introduction

Today computer networks are everywhere. In our everyday life we are almost always connected to the Internet and since many business critical applications also need network connection in most of the cases we are also using them in our working hours. The different demands of heterogeneous networks has led to a situation where nowadays IP networks are very complex to both build and manage. The current network architectures are very rigid thus it is especially very hard to implement new features into them.

Software Defined Networking (SDN) offers a solution for this problem with mainly the following features: i) it decouples the data and control plane thus network devices become simple forwarding elements (also called SDN switches), ii) control logic is moved out to an external Network Operating System (also called the SDN controller) which makes the decisions about forwarding rules and communicates them to the SDN switches via open protocol standards (e.g. OpenFlow), 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 [5], packet delay [5], [7] and traceroute [8] measurements. We couldn't find any paper in the 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 provider frequently use them for network management and traffic engineering purposes. Furthermore, nowadays, video streaming generates the biggest portion of Internet traffic where ABW techniques plays a significant role in adopting to the current network load.

The main contribution of this paper is to present a method for available bandwidth measurement in Software Defined Networks. Taking advantage of the features that SDN offers, we present a solution for this problem in three different networking scenarios: i) when paths in the network are fixed thus we want to know the route between two points and the associated available bandwidth, ii) when paths are not fixed thus we want to find the best possible available bandwidth between to points in the network and the associated route, iii) the same scenario as the previous one in multipath environment thus 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].

The reminder of this paper is structured as follows. Section II presents the background of Software Defined Networks and the related work. In Section III we present our approach for measuring available bandwidth in SDN. Section IV describes the test configuration that we used to validate our method. The results of tests are presented in Section **??**. Finally, in Section V we conclude our work.

II. Background and Related Work

Although Software Defined Networking only gained significant focus by the research community after the introduction of OpenFlow [1], the main concepts of SDN root in earlier works in the fields of active networks, control and data plane separation and network virtualization [11]. In this paper we follow the definition of SDN as presented in [12] which is based on the following four elements.

- Control and data planes are separated from each other. Network devices no longer have control functionalities,

they become simple forwarding devices.

- Forwarding rules are made based on a set of fields in the packet headers (not by the destination of the packet). This also guarantees unified behaviors of networking elements such as switches, routes or firewalls.
- The control plane is moved to an external entity called the Network Operating System (NOS) or SDN controller. NOS is a software that runs on commodity hardware and it can communicate the forwarding rules to the switches via open standards.
- Third party application can program the network over the NOS. The controller must also provide the necessary abstractions and interfaces for serving these applications.

Fig. 1 presents the architecture of Software Defined Networks. The SDN controller can communicate with the switch via the southbound API. Arguably, the most used standard for southbound API is OpenFlow, but there are other proposals for e.g. OVSDB [13], POF [14] or ROFL [15]. For NOS platform there are many available open softwares such as NOX [16], POX [16], Floodlight [10] or Ryu [17]. Moreover, there are ongoing industrial consortia projects for data center specialized controller platforms, for e.g. OpenDayLight [18] or OpenStack [19]. As shown if Fig. 1, SDN applications can program the network using the northbound API of the NOS. However, these APIs are specific to controller software thus most of the currently available SND applications are only able to work over one NOS platform.

In the recent years, there has been several proposals for monitoring Software Defined Networks. FlowSense [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 [6] offer adaptive sampling algorithms that can adopt for the current network load. However, their approach 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 [5] also offers solution for packet loss and delay monitoring as well. For packet loss measurement it polls the flow counters in the ingress and egress switches of a given flow and calculates the difference. For delay measurement it uses the SND controller to send packet probe packets to the network along a given path and than looped back to the controller. Using the round trip times to the ingress and egress switches the tool is able to calculate the delay for the given path. Phemius and Bouet [7] also use the same approach for delay measurement but they observed a constant difference between the measured and reference time values. They also present a method for calculate this value and calibrate the delay measurement according to this calculation.

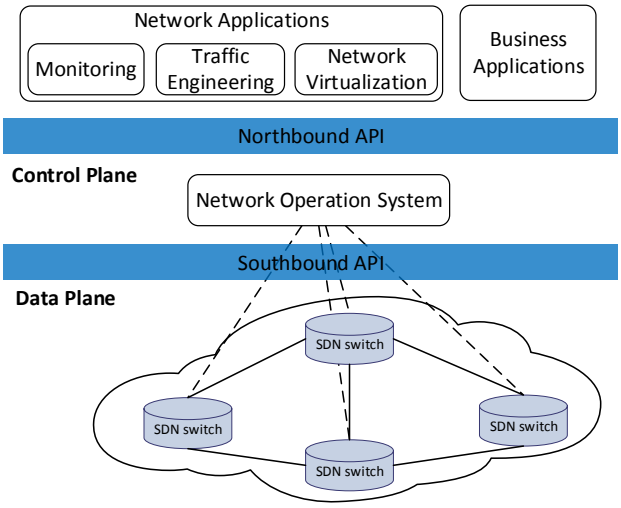

Fig. 1: The architecture of Software Defined Networks

We found that the current literature lacks of a solution for measuring available bandwidth in SDN. Furthermore, a comprehensive taxonomy of different elements in Software Defined Networking can be found in [12].

In classical networks available bandwidth techniques are classified into active and passive techniques. Passive methods use multiple measurement points in the network to monitor the bandwidth utilization, the packet loss ratio and the packet delay. With the synchronization of this measurement one can estimate the available bandwidth. However, these methods are very complex to deploy thus they are rarely used in practice. The active methods operate by sanding probe packet to the network. Most commonly used tools like Iperf [20] or Ookla [21] use a single TCP flow to saturate the network path. Since this method suppresses other traffic on the network they are rarely used for professional purposes.

Active ABW methods in the literature is usually categorized into packet gap and packet rate models. Packet gap tools such as Spruce [22] or Traceband [23] use packet pair to calculate the ABW. Based on the difference of the entry and the exit time gap of the packet pairs they are able to estimate the load on the bottleneck link, thus along with the bottleneck link capacity they are able to calculate the available bandwidth. On the other hand, packet rate tools use multiple series of packet with different rates. They iteratively increase the sending rate until congestion occurs thus they are able to estimate the available bandwidth without any knowledge of the network. Examples for packet rate tools include PathLoad [24] and PathChrip.

However, most of the ABW tools that are currently available can only work in certain networking scenarios after precise calibration [25]. In our approach we use a passive available measurement method by taking advantage of the NOS on the architecture of SDN. We use the northbound API of the Floodlight controller to discover the topology of the network and to monitor the bandwidth utilization of the links. With this

TABLE I: Notation list

Notation	Description
$G(V, E)$	the directed graph representation of the network topology with node set V and edge set E
e_i	i^{th} link in the network topology graph
c_i	the capacity of e_i
b_i	the current bandwidth load on e_i
a_i	the available capacity on e_i, $a_i = c_i - b_i$
$P_{A\to B}$	the set of all available paths from A to B

information we are able to calculate the available bandwidth for any path in the network in any given time.

III. Measuring Available Bandwidth in SDN Networks

In our available bandwidth application we take advantage of the network abstractions provided by the NOS. Using northbound API of the SDN controller we are able to query all the switches operating in the network and 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 (for further notations see Table 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 similar approach that was previously presented in papers [4]–[6] thus we periodically poll the counters in the SDN switches using the *FlowStatsReq* OpenFlow message. This method is already proven to be working effectively in SDN and it provides an easy solution for measuring the bandwidth utilization over the entire network. After this step, we are able to calculate the *available capacity* a_i on every link in the network. Based on the a_i values we are able to calculate the available bandwidth on a given P path by the following formula

$$ABW_P = \min_{e_i \in P} a_i. \tag{1}$$

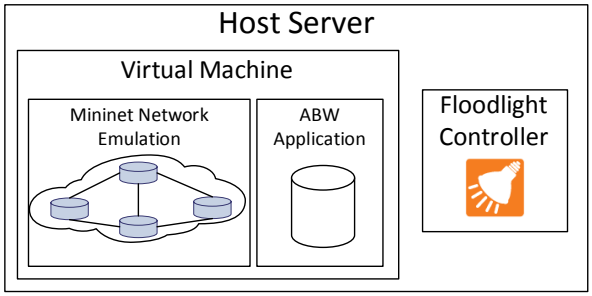

Fig. 2: The assembled test configuration

The application is also able distinguish three different scenarios and calculate the ABW according to it. These cases are the following.

1) **ABW on fixed paths.** In this scenario the routing policies are fixes thus for a given flow first we have to find out it's route on the network and than calculate the available bandwidth using Eq. (1). For this the application can use the northbound API of the NOS, e.g. Floodlight's REST based API provides an interface for telling the route of a flow in the network (with any given headers on a given entry point) according to the policies set up in the controller.
2) **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 expressed by the following equation:

$$ABW_{A\to B} = \max_{P \in P_{A\to B}} \min_{e_i \in P} a_i. \tag{2}$$

For solving this equation we use a modified Dijstra algorithm where the metric of a path is not measured by the sum of the edges' capacities (distances) but by Eq. (1). This algorithm also gives the best possible path for the best AWB solution in $O(|E| + |V| \log |V|)$ (just like a normal shortest-path Dijstra algorithm would).
3) **Multipath scenario.** In this case we can use multiple paths between two points for flowing the traffic 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 [26]. In this case we face off a classical max-flow problem over the network topology graph $G(V, E)$ which can be solved by the Ford-Fulkerson Algorithm in $O(|E|f)$ complexity (where f is the maximum flow in the graph).

IV. Test Configuration

Fig. 2 presents the schematics of the assembled test configuration. 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 Table II.

The ABW application also runs in the Mininet virtual machine. The reason for that is that parallel with the ABW calculation method presented in Section III, we also use a kernel polling mechanism for generating reference values for the measurements. Mininet creates separate network interfaces in the Linux system for every interface of the emulated SDN switches. This allows us to use IPtables for ground truth data generation since this method provides more accurate measurement of the traffic on the interfaces. Packets between

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

TABLE II: 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

the ABW application and the Floodlight controller can (and in some of our test cases will intentionally) suffer variable delay thus this method can only approximate the actual traffic.

Fig. 3 sketches the network topology we created in Mininet for testing our AWB application. S1, S2 and S3 creates a classical Y topology which is frequently used as a testbed for testing ABW applications [25]. The idea is to set up the link between S1 and S2 to serve as the bottleneck link (lowest capacity on the route) 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 tight link will become different which can interfere the calculations of current ABW tools [25]. 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 polling application and the Floodlight controller.

The default route policy in Floodlight won't 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 that's not the default one. If the cross traffic from H3 to H4 is larger than 10 Mbps than the alternative route through S4 would provide a better path with larger available bandwidth.

Furthermore, we use D-ITG [27] for traffic generation which was proven to work much reliable than other traffic generation platforms [28]. Using D-ITG, we are able generate constant bit rate traffic with very precise inter departure times, variable bit rate traffic using different stochastic distributions or replay traffic according to previously captured traces. D-ITG also uses logging mechanism for further reference tracking of the packet inter departure times. Moreover, hosts are configured with the CPU isolation method presented in Mininet HiFi [29] thus they can't interfere the traffic generation process of each other.

Possible scenarios to present:

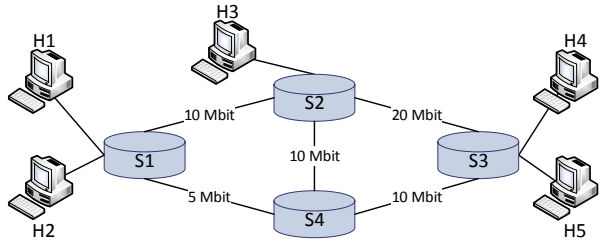

Fig. 3: The test topology in Mininet

1) Static scenrio with some CBR traffic on the net.
2) VBR traffic using Pareto distribution.
3) VBR traffic using variable delay.
4) Long term scenario using real traffic pattern.

V. Conclusion

In this report we presented an application that is able to measure end-to-end available bandwidth (ABW) in Software Defined Networks (SDN). Our application uses the Network Operating System (NOS) to generate a graph representation of the network topology and then use OpenFlow messages to track the bandwidth utilization over every link in the network. Based this information, we are able to calculate the ABW on every path in the network. We also assembled a test environment using Mininet for network emulation and Floodlight for SDN controller in order to test our application in various network conditions. Our results show that using the presented technique for ABW measurement in SDN is satisfactory for both short and long time scale measurements.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Computer Communnication Review*, vol. 38, no. 2, pp. 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*, ser. Lecture Notes in Computer Science, 2013, vol. 7799, pp. 31–41.

[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)*, Nov 2013, pp. 41–46.

[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 (NOMS)*, May 2014, pp. 1–9.

[5] 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*, May 2014, pp. 1–8.

[6] 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.

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

[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*, 2014, pp. 145–150.

[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*, 2010, pp. 19:1–19:6.

[10] "Floodlight," retrieved: March, 2015. [Online]. Available: http://www.projectfloodlight.org/

[11] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, April 2014.

[12] 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*, vol. 103, no. 1, pp. 14–76, Jan 2015.

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

[14] 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*, ser. HotSDN '13, 2013, pp. 127–132.

[15] 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)*, Sept 2014, pp. 123–124.

[16] "NOX and POX SDN Controllers," retrieved: March, 2015. [Online]. Available: http://www.noxrepo.org/

[17] "RYU network operating system," retrieved: March, 2015. [Online]. Available: http://osrg.github.com/ryu/

[18] "OpenDayLight Project," retrieved: March, 2015. [Online]. Available: http://www.opendaylight.org

[19] "OpenStack Project," retrieved: March, 2015. [Online]. Available: https://www.openstack.org/

[20] "Iperf," retrieved: March, 2015. [Online]. Available: http://sourceforge.net/projects/iperf/

[21] "Ookla Speedtest," retrieved: March, 2015. [Online]. Available: http://www.speedtest.net/

[22] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proc. ACM SIGCOMM conference on internet measurements*, Oct. 2003, pp. 39–44.

[23] C. D. Guerrero and M. A. Labrador, "Traceband: A fast, low overhead and accurate tool for available bandwidth estimation and monitoring," *Computer Networks*, vol. 54, no. 6, pp. 977–990, 2010.

[24] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput," *IEEE/ACM Transaction on Networking*, vol. 11, no. 4, pp. 537–549, Aug. 2003.

[25] A. Botta, A. Davy, B. Meskill, and G. Aceto, "Active techniques for available bandwidth estimation: Comparison and application," in *Data Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science, 2013, vol. 7754, pp. 28–43.

[26] B. Sonkoly *et al.*, "Sdn based testbeds for evaluating and promoting multipath tcp," in *Proc. IEEE International Conference on Communications (ICC 2014)*, June 2014, pp. 3044–3050.

[27] A. Botta, A. Dainotti, and A. Pescapé, "A Tool for the Generation of Realistic Network Workload for Emerging Networking Scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531 – 3547, 2012.

[28] ——, "Do you trust your software-based traffic generator?" *IEEE Communications Magazine*, vol. 48, no. 9, pp. 158–165, Sept 2010.

[29] 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)*, 2012, pp. 253–264.