Estimation of End-to-End Available Bandwidth and Link Capacity in SDN

Manmeet Singh(✉), Nitin Varyani, Jobanpreet Singh, and K. Haribabu

Department of Computer Science and Information Systems, BITS, Pilani, Pilani Campus, Pilani, India
{f2012763,f2009586,f2012124,khari}@pilani.bits-pilani.ac.in

Abstract. The traditional networks, with the control and data plane integrated into the same network devices, do not provide a global view of the network performance like degree of congestion, bandwidth utilization, etc. Software defined network (SDN) is an approach towards this problem which separates the control plane of the switch from its data plane and provide a centralized control plane so as to get a global view of the network performance and thus make decisions of how to regulate flows. In SDN, network monitoring can be achieved more efficiently than traditional networks using OpenFlow statistics. SDN controller can keep track of available bandwidth on each link and thus estimate end-to-end available bandwidth of a path simply by composing individual link bandwidths thus avoiding end-to-end probing. We have made two contributions in this paper: (i) proposed and validated a method to estimate end-to-end available bandwidth on any given path by composing link-wise available bandwidths (ii) proposed a method to measure link capacity using OpenFlow protocol. We compared our results to the ones obtained using the state-of-the-art bandwidth measurement tool, Yaz.

Keywords: Link capacity · End-to-end available bandwidth
SDN controller

1 Introduction

Software Defined Networking (SDN) aims to effectively program the network with software running on a central controller. Today's network switches and routers program their forwarding tables locally, which means that network devices make their own decisions internally about how to forward traffic. Traffic-forwarding decisions are informed by distributed control-plane protocols like spanning-tree, OSPF and BGP. But these traditional networking protocols have limited flexibility. In order for them to work, all network devices participating in the forwarding domain have to follow the same rules as defined by the protocol standard. That leaves little room for creativity or unusual business requirements.

M. Singh and N. Varyani—Both authors contributed equally to this work.

N. Kumar and A. Thakre (Eds.): UBICNET 2017, LNICST 218, pp. 130–141, 2018.
https://doi.org/10.1007/978-3-319-73423-1_12

SDN [1,2] is an emerging networking paradigm that overcomes the limitations of current network infrastructures. In case of traditional networks, we have both the control plane and the data plane integrated into the same network devices. SDN comes up with an approach to break this vertical integration by separating the control logic from the underlying routers and switches. With the separation of the control and data planes, network switches performs only the forwarding operations based on control logic which is implemented in a logically centralized controller (or network operating system), simplifying policy enforcement and network (re)configuration and evolution [3]. A logically centralized programmatic model does not postulate a physically centralized system [4]. The need to achieve adequate levels of performance, scalability and reliability would resist such a solution. Instead, production-level SDN network designs resort to physically distributed control planes [4,5]. To achieve the separation between the control plane and the data plane we have a well implemented programming interface between network devices (like switches and routers) and the SDN controller. The controller maintains the global view of the underlying network and the applications can obtain the statistics of the network state for improved performance. Also, the SDN controller controls the state of the dataplane elements with the help of well-defined communication protocol. OpenFlow [1,2] is a protocol for communication between the control and the forwarding layers of an SDN architecture. OpenFlow enabled switches provide important statistics about the network like port statistics, flow statistics, packets transmitted, packet received and packet loss.

The statistics obtained using SDN Controller can be used to estimate end to end available bandwidth. The end-to-end available bandwidth is defined as the maximum rate that the path can provide to a flow, without reducing the rate of the traffic in that path. This information can be very useful in congestion control, streaming applications and network selection. Our paper suggests an approach to estimate and validate the end-to-end available bandwidth. We have also proposed a method to estimate the capacity of links using statistics obtained from SDN Controller. We have validated the end-to-end available bandwidth values using state-of-the-art tools for bandwidth estimation like Yaz. We have also analyzed the effect of change in the polling time interval, the time interval after which SDN controller requests the network statistics, on the estimated end-to-end available bandwidth.

The reminder of the paper is structured as follows. Section 2 presents a short background on the related work. In Sect. 3 we present our design approach for measuring end-to-end available bandwidth in SDN. Sect. 4 describes the experimental setup we used to validate our method and Sect. 5 discusses about the results of tests. Finally, Sect. 6 ends the paper with concluding remarks.

2 Related Work

There are several end-to-end network performance measurement tools proposed in the literature such as Spruce [1], Pathload [3], IGI/PTR [4], Abing [5], pathChirp [5], DietTopp [6], Yaz [7], and ASSOLO [5]. These end-to-end measurement tools use packet pair techniques to measure performance metrics like bandwidth, delay, latency, etc. These techniques need to send packets from one side in a particular pattern and receive on the other end and analyze the arrival time etc. These approaches are required to be carried out on end-to-end basis. These techniques lead to packet duplication if bandwidth values are estimated for multiple end hosts. This will also cause congestion in network. This may also lead to increase in packets drop rate because of which estimated bandwidth values might not be accurate. All these limitations can be easily overcome by the use of SDN controller which queries network statistics and can estimate end-to-end available bandwidth for any path without packet duplication. Not only this, controller can also set high priority to the packets sent for bandwidth estimation so that these packets are not dropped on the way leading to more accurate bandwidth values than the existing methods.

OpenNetMon [8] is an existing approach in SDN to monitor per-flow metrics like throughput, delay and packet loss between source and destination. This approach measures performance metrics for each flow separately and not for a given path. Our proposed approach measures available bandwidth link-wise and composes on-demand to find out end-to-end available bandwidth for a given path. In [9], the authors measures the end-to-end available bandwidth between any two end-hosts in the network using SDN statistics but their work assumes link capacity to be known in advance. Our approach measures the link capacity of the links dynamically without having to run sender or receiver applications on end hosts. Additionally, we compare our results with well-known tool for bandwidth estimation in order to validate our results.

3 Design Approach

Our approach learns the topology using the controller API. Using this topology, it queries the statistics from the switches in the topology. These statistics are used to estimate the available bandwidth on each link in the network. Instead of using static values for the maximum capacity, we are estimating the capacity dynamically. This is because the links which are visible to the controller are not often the physical links. They may be aggregated links. The Fig. 1 given below illustrates our approach. The module "Create Topology" is storing the network topology fetched from SDN controller and is regularly updated. The module "Store Port Statistics" stores transmitted bytes for each port in a switch and is updated more frequently. This module is also storing time elapsed. The module "Estimate Link Capacity" estimates the link capacity dynamically.

3.1 Learning Topology and Consumed Bandwidth Using Controller

A hashable data structure is used to store the topology and consumed bandwidth/data rate in each link. A pair (S, D) is used as a key which is mapped to the ordered set $(P_1, P_2, TB, T, DR, MB)$ where S and D are source and destination mac addresses respectively of two switches connected directly in the network. Source and destination switches are connected through each other using their

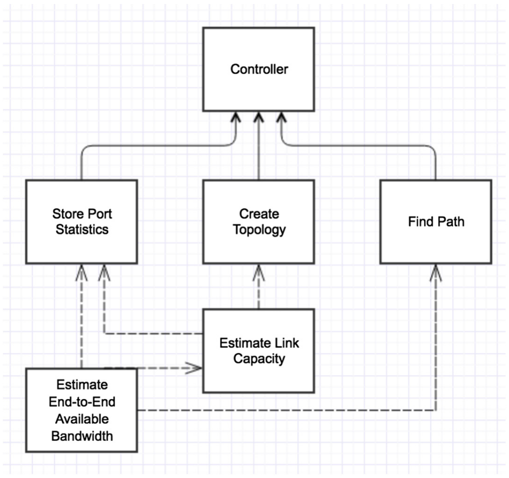

Fig. 1. Diagram illustrating our design approach

respective port P_1 and P_2. DR represents the bandwidth consumed between source and destination switch, T represents the time elapsed, TB is the bytes transmitted by the source S to destination D from beginning to time T and MB is the maximum bandwidth (capacity) of the link connecting the two switches which is estimated dynamically using the method described in Sect. 3.4. We then calculate the consumed bandwidth/datarate(DR) in the path from S to D at time T_{new} using the formula

$$DR = (TB_{new} - TB_{old})/(T_{new} - T_{old}) \tag{1}$$

where TB_{new} denotes the number of bytes transmitted through port P_1 since beginning to time T_{new}, TB_{old} is the number of bytes transmitted through port P_1 since beginning to time T_{old}. Using the Open Daylight statistics, an observation was made that for two switches connected by a link, the received bytes by one switch was much more that the bytes transmitted by the other switch at the ports through which they are connected. Such discrepancy is resolved by replication of packets from one interface of switch to all another interfaces within the switch when it receives a packet. Thus, to have proper calculation of bandwidth available in links, we have used transmitted bytes.

Same data structure is used to store the information regarding which switch is connected to which end host along with the consumed bandwidth in those links.

3.2 Finding Path Between Source and Destination

For a given flow, we are finding its route on the network set by the controller, and then calculating its end-to-end available bandwidth. We use the northbound API provided by the SDN controller for fetching the route of a flow in the network set by the controller according to the policies.

3.3 Estimating Available Bandwidth Between Source and Destination

For each link, we subtract Data Rate (DR) from Maximum Bandwidth (MB) to get available bandwidth of the link. The minimum of the available bandwidths of links along a path is the end-to-end available bandwidth of the path. Our approach estimates end-to-end available bandwidth by composing link-wise bandwidth which avoids generation of redundant traffic if bandwidth values have to be estimated for multiple pair of hosts.

3.4 Estimating Link-Capacity Dynamically

We need the value of the maximum available capacity in order to estimate the available bandwidth. There are many available Opensource tools which can find the capacity of the path. Pathrate [10] estimates the bottleneck capacity by sending packet pairs, called probing packets, back-to-back and measuring the dispersion of the packet pairs. Dispersion varies among different packet pairs and the packet pair bandwidth distribution is analyzed to relate it with capacity.

But with these tools, we need to have receiver and sender applications running on the end hosts. In our approach, the controller itself controls the traffic and uses the port statistics to measure the capacity of a given link. The controller generates UDP packets with a unique destination port number. Since the network topology does not change so frequently, the measurement of capacity need not be so frequent, and thus it will not significantly affect the real traffic. We added flow entries in the ovs-switches to direct the traffic towards the link for which we need to measure the capacity. Consider two ovs-switches s1 and s2 connected with a link l whose capacity needs to be estimated. We add a flow

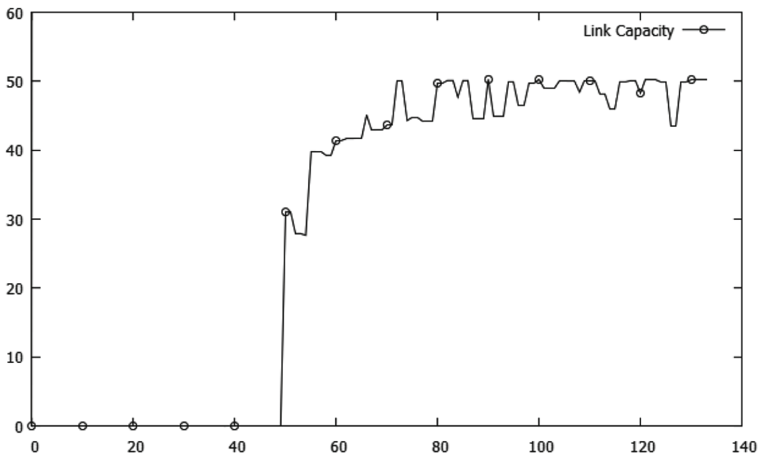

Fig. 2. Estimating link capacity of 50 MB link

entry in s1 to forward all the packets with the predefined unique UDP destination port number P to s2. In s2, we add a flow entry to drop the packet with the unique port number P as these packets are just used for measurement of capacity. We measure the rate at which data is received at s2 using port statistics. As shown in Fig. 2, the traffic is increased in order to estimate the maximum rate at which data is received at the receiver end of the link. We stop iterating once we do not observe any change in the maximum value of received throughput, which is the final link capacity.

4 Experimental Setup

4.1 Bandwidth Estimation Module

As discussed in previous section, we estimate the available bandwidth between the source and the destination by composing link-wise bandwidths using statistics obtained from the controller. The bandwidth values will be used by the validation module to analyze its correctness by comparing it with a well-known bandwidth measurement tool, Yaz.

4.2 Validation Module

This module takes care of analyzing the difference in the bandwidth values obtained from our work and other available tools. The first measurement is the value given by our SDN script for bandwidth measurement. The second measurement is the bandwidth value as given by an open source bandwidth measurement tool, Yaz. While the Yaz adopts heuristic methods for estimating bandwidth by considering the delay between packet streams, our method uses port statistics like transmitted bytes in a given link to estimate the bandwidth.

The basic difference between Yaz and our method is that Yaz uses an experimental approach, sending packet probes and observing delays to measure the available bandwidth but our approach uses a statistical approach, using the data of all switches in the path to find out the end-to-end available bandwidth.

4.3 Traffic Generation Module

The tool used for generating regulated traffic is Distributed Internet traffic generator (D-ITG). D-ITG can be used to generate traffic in a network with controllable parameters like packet rate, packet size, etc. DITG can be used to generate traffic in a particular distribution like normal or Poisson distribution.

4.4 Link Capacity Estimation Module

As discussed in Sect. 3.4, we need to know the capacities of the links in the path. We estimated the capacity of mininet links for various topologies and the results are given in Table 1. We have also estimated the capacity of the physical LAN wire used in our setup in Fig. 5. The actual capacity of the LAN wire used is 100 Mbps. The first row in Table 1 corresponds to the physical LAN wire.

The estimated values (in Table 1) are higher than expected values by a small amount in all the experiments dues to some experimental errors.

Table 1. Expected capacity and estimated capacity

Expected capacity (Mbps)	Estimated capacity (Mbps)
100	101.2344456787
50	50.2809047531
20	21.4398119679
10	10.8275808985

5 Results and Discussion

5.1 Results on Mininet Testbed

In our first setup, OpenDaylight was used as a controller and mininet was used to emulate the topology. A custom topology was created in python for our experimentation (Fig. 3). 10.0.0.11 is the IP address allocated by mininet to the physical computer in which mininet, opendaylight and library is running and can be seen in the Fig. 3. Mininet network was connected to the physical computer using Network Address Translation (–nat).

A cross traffic was generated in the network using iperf tool. At node 10.0.0.4, a user application is connected to our library and is requesting our library end-to-end available bandwidth between 10.0.0.4 and 10.0.0.6. As can be observed

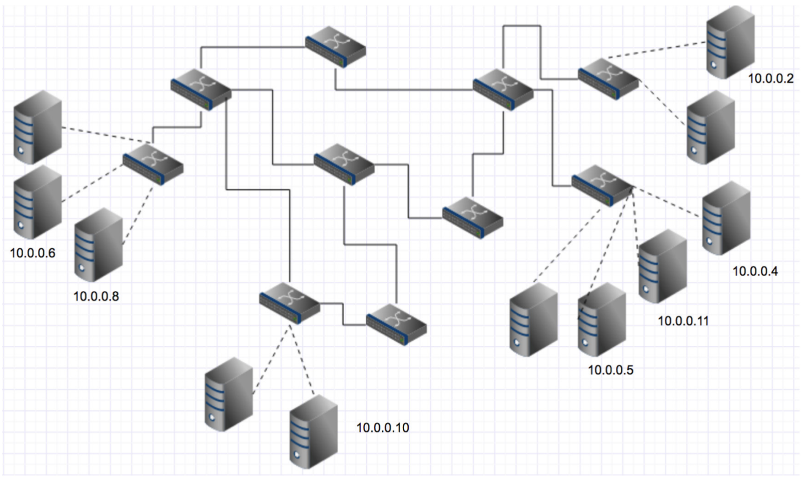

Fig. 3. Mininet testbed

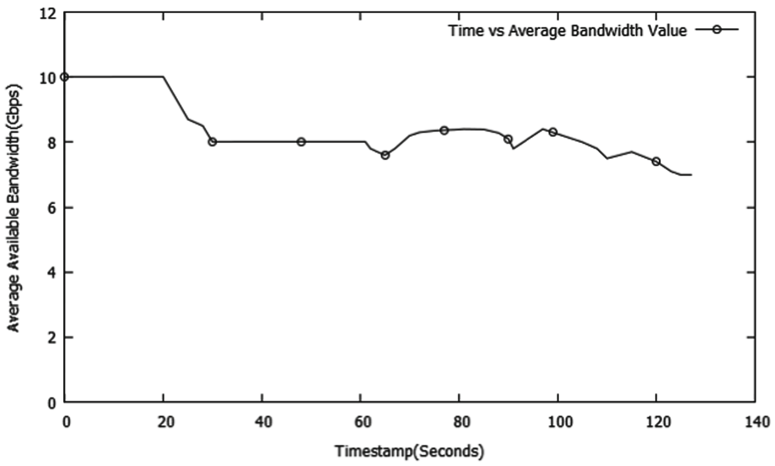

Fig. 4. Time vs. end-to-end available bandwidth between 10.0.0.4 and 10.0.0.6

from Fig. 4, initially there was no traffic in the network and hence we can observe the value of 10 Gps bandwidth available between 10.0.0.4 and 10.0.0.6.

We can observe a dip at t = 20 s. This is due to udp traffic of 350 Mbps sent from 10.0.0.6 to 10.0.0.4. Then a udp traffic of 350 Mbps from node 10.0.0.10 to node 10.0.0.2 was added to the network at around 60 s. We can observe that the traffic has become turbulent. Another udp traffic of 350 Mbps from 10.0.0.8 to 10.0.0.5 was added to the network at around 100 s. We can thus observe a dip in the bandwidth available at 100 s.

5.2 Results on Physical Testbed

We created a second testbed which involved a physical link between the switches to validate our approach in this scenario also. We have created a testbed which consists of two OVS switches, each installed on a separate physical machine (Host-2 and Host-4 in Fig. 5). These two machines, and hence the switches, are connected by a LAN wire of capacity 100 Mbps. Each physical machine has a Virtual Machine (VM) acting as a host (Host-1 and Host-3 in Fig. 5). Each of these VMs are connected to the switches as shown in Fig. 5. Hence we have a linear topology with two switches and four hosts.

We are using D-ITG for the generation of traffic inside the network. We have used ryu controller for this testbed.

5.3 Comparing Results with Yaz

Figure 6 shows the end-to-end available bandwidth values obtained from Yaz and our application plotted against time-stamp. We can observe significant amount of similarity between Yaz results and our work.

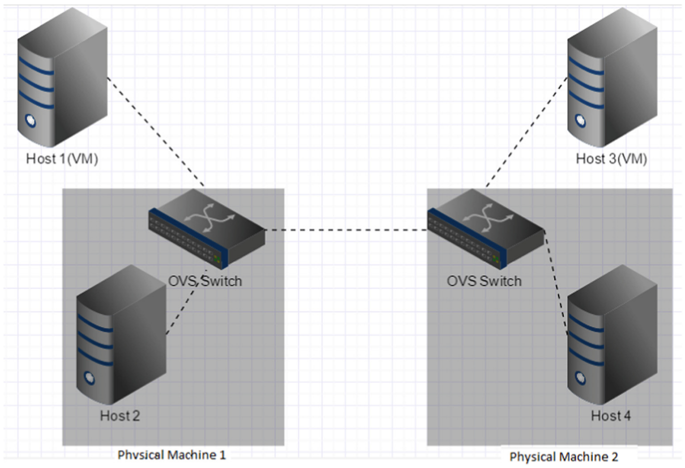

Fig. 5. Testbed with physical link and ovs-switches

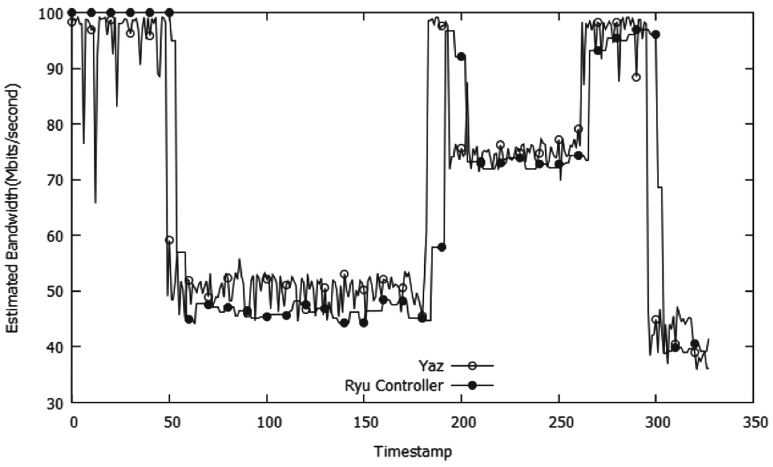

Fig. 6. Comparing the bottleneck bandwidth of the obtained path measured using Yaz and our method

During the experiment, the amount of traffic generated through the link was varied after every few seconds so as to verify that irrespective of varying the traffic through the link, the values given by our Ryu script and Yaz tool are comparable.

The accuracy α in this experiment is computed as:

$$\alpha = 1 - |(B_{yaz} - B_{ryu})/B_{yaz}| \qquad (2)$$

where B_{yaz} and B_{ryu} is the bandwidth obtained by yaz and ryu respectively. The graph shows that the bandwidth calculated using our approach is very close to the actual bandwidth available, as calculated by the tool yaz. These two measurements come from two methods which use an entirely different approach from each other, and yet give comparable results and thus provides a ground for the correctness of our approach. The advantage of this statistical method over the experimental method of Yaz is that in this case we have the available bandwidth corresponding to each link in the path which gives us the knowledge of which link in the path has the minimum available bandwidth. Another advantage of this method is that it saves time by storing link wise bandwidth which can be used instantaneously to calculate bottleneck bandwidths for multiple paths. Unlike in heuristic methods for bandwidth estimation, we do not need to do end to end probing in our approach which takes a minimum of round-trip time.

5.4 Estimating Bandwidth Values with Different Polling Time Intervals

One of the important parameters in our approach is the time after which port statistics are requested for every switch from the controller at regular intervals. It is important to note that in a large network, it will put significant pressure/computing delay on the network if we request stats from the controller at a high frequency. Therefore it is important that this time gap between two such requests for statistics is tuned according to the size of the network. We ran different experiments where we varied the polling time interval. We then calculated the cumulative of the end-to-end available bandwidth measured by Yaz and our application across time. In Fig. 7, the absolute of the difference of the two cumulative values was plotted on the y-axis with the time on the x-axis. From the figure, we can infer that the difference increases with increase in polling time. This is because as we increase the polling time, the bandwidth

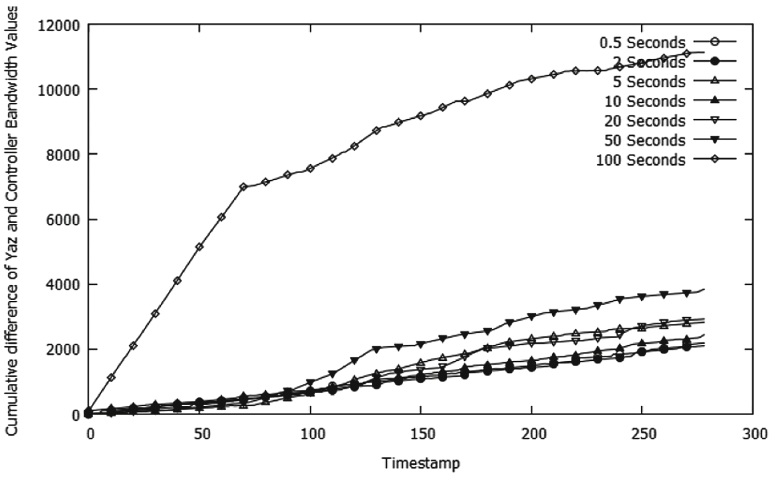

Fig. 7. Cumulative absolute difference between throughput of Yaz and Ryu

estimation period will also be increased and we may not be able to capture the accurate bandwidth with that time delay. But, with the polling time interval of 500 ms, the difference in cumulative bandwidth values is smaller and more closer to the cumulative bandwidth value of Yaz. Thus, for our testbed, we found the polling time of 500 ms to give most accurate measurement of end-to-end available bandwidth.

6 Conclusion

In this paper we present an approach to measure end-to-end available bandwidth in Software Defined Networks (SDN). We have also validated our results with a well known bandwidth estimation tool, Yaz. The bandwidth calculated by our approach is very close to the actual bandwidth available, as calculated by the tool Yaz. We have also presented an approach to find the maximum capacity of the link dynamically rather than taking static values. We also observed the impact of controller polling time interval on the estimation of bandwidth.

There are few areas which are still undiscovered like impact of our approach on the network traffic. SDN controller keeps on querying statistics from all the switches in network after fixed interval which may lead to excessive traffic. We may improve upon this in our future work by querying only the selected switches based on some factors like ignoring stable switches in non-turbulent environment. We may also work on the understanding the scalability of our approach by considering larger networks. The proposed solution doesnt require any application running on end hosts and can be used by various applications to tune the parameters based on the available bottleneck bandwidth.

References

1. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
2. ONF: Open networking foundation (2014). https://www.opennetworking.org/
3. Kim, H., Feamster, N.: Improving network management with software defined networking. IEEE Commun. Mag. **51**(2), 114–119 (2013)
4. Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., Shenker, S.: Onix: a distributed control platform for large-scale production networks. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI 2010, p. 16. USENIX Association, Berkeley (2010)
5. Navratil, J., Cottrell, R.L.: ABwE: a practical approach to available bandwidth. In: Proceedings of the 4th International Workshop on Passive and Active Network Measurement PAM 2003 (2003)
6. Jain, M., Dovrolis, C.: Pathload: a measurement tool for end-to-end available bandwidth. In: Proceedings of the 3th International Workshop on Passive and Active Network Measurement PAM 2002 (2002)
7. Hu, N., Steenkiste, P.: Evaluation and characterization of available bandwidth probing techniques. IEEE JSAC **21**(6), 879–894 (2003)

8. Van Adrichem, N.L.M., Doerr, C., Kuipers, F.A.: OpenNetMon: network monitoring in openflow software-defined networks. In: 2014 IEEE Network Operations and Management Symposium (NOMS). IEEE (2014)
9. Megyesi, P., et al.: Available bandwidth measurement in software defined networks. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. ACM (2016)
10. Dovrolis, C., Ramanathan, P., Moore, D.: Packet-dispersion techniques and a capacity-estimation methodology. IEEE/ACM Trans. Netw. **12**(6), 963–977 (2004)
11. Rechert, K., McHardy, P., Brown, M.A.: HFSC Scheduling with Linux
12. Strauss, J., Katabi, D., Kaashoek, F.: A measurement study of available bandwidth estimation tools. In: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement IMC 2003 (2003)
13. Ribeiro, V., Riedi, R., Baraniuk, R., Navratil, J., Cottrell, L.: pathChirp: efficient available bandwidth estimation for network paths. In: Proceedings of the 4th International Workshop on Passive and Active Network Measurement PAM 2003 (2003)
14. Johnsson, A., Melander, B., Bjorkman, M.: DietTopp: a first implementation and evaluation of a simplified bandwidth measurement method. In: Proceedings of the 2nd Swedish National Computer Networking Workshop (2004)
15. Goldoni, E., Rossi, G., Torelli, A.: Assolo, a new method for available bandwidth estimation. In: Proceedings of the Fourth International Conference on Internet Monitoring ICIMP 2009, pp. 130–136, May 2009
16. Luckie, M.J., McGregor, A.J., Braun, H.-W.: Towards Improving Packet Probing Techniques Science (1989)
17. Pakzad, F., Portmann, M., Hayward, J.: Link capacity estimation in wireless software defined networks. In: 2015 International Telecommunication Networks and Applications Conference (ITNAC). IEEE (2015)