Computer Communications 99 (2017) 48–61

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Challenges and solution for measuring available bandwidth in software defined networks

Péter Megyesi[a], Alessio Botta[b], Giuseppe Aceto[b], Antonio Pescapé[b,*], Sándor Molnár[a]

[a] Budapest University of Technology and Economics, Budapest, Hungary
[b] University of Napoli Federico II, Naples, Italy and NM2 srl, Italy

ARTICLE INFO

Article history:
Received 11 June 2016
Revised 12 October 2016
Accepted 3 December 2016
Available online 8 December 2016

Keywords:
SDN
OpenFlow
Floodlight
OpenDaylight
ONOS

ABSTRACT

Software Defined Networking (SDN) is an emerging paradigm that is expected to revolutionize computer networks. Methods for measuring Quality of Service (QoS) parameters such as bandwidth utilization, packet loss, and delay have been recently introduced in literature for SDN-based scenarios, but they required almost invariably a completely different approach with respect to traditional network environments, thus facing new challenges and exploiting new opportunities. An important dynamic path characteristic is Available Bandwidth (ABW), that has strong impact on a wide range of applications, but is a metric very hard to estimate with traditional approaches. In this paper we focus our analysis on ABW measurement based on messages in the OpenFlow protocol. We present both analytical results and experimental evaluation (in Mininet emulation and using Floodlight, OpenDaylight and ONOS controllers) of measurement error due to network delay between the SDN switches and the controller. Based on our results we propose to extend the OpenFlow protocol with a local timestamping mechanism, providing and discussing two different implementations of this feature. The presented analysis and the proposed extension of OpenFlow protocol are not restricted to ABW, and can benefit measurement of other network metrics in SDN.

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, several proposals for monitoring Quality of Service (QoS) parameters in SDN networks have been presented in literature. They mostly tackle problems related to bandwidth utilization [2–6], packet loss ratio [5], packet delay [5,7], and route tracing [8]. All these monitoring solutions are based on approaches completely different from the counterparts in traditional networks, and this is mainly due to the abstraction mechanism provided by the Network Operating System (NOS). However, the new possibilities provided by SDN and its NOS introduce new issues, limitations, and sources of error, which were previously undiscussed in such manner.

The main contribution of this paper is four-fold. Firstly, we present the state-of-the-art Available Bandwidth monitoring techniques used in Software Defined Networks emphasizing how they utilize the new features introduced by the architecture. Secondly, we discuss the limitation of such monitoring approaches and the new source of errors they introduce, with analytical calculation of measurement error due to lack of local timestamping mechanism in OpenFlow. Thirdly, we validate experimentally in Mininet emulation testbed the analysis and the properties of the proposed technique (using the controllers Floodlight, OpenDaylight and ONOS).

* Corresponding author.
E-mail addresses: megyesi@tmit.bme.hu (P. Megyesi), a.botta@unina.it (A. Botta), giuseppe.aceto@unina.it (G. Aceto), pescape@unina.it (A. Pescapé), molnar@tmit.bme.hu (S. Molnár).

http://dx.doi.org/10.1016/j.comcom.2016.12.004
0140-3664/

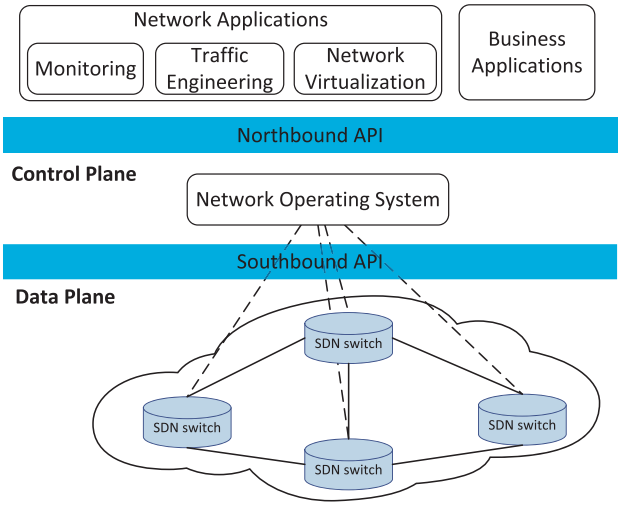


Fig. 1. The architecture of Software Defined Networks.

Finally, we propose an extension to the OpenFlow protocol providing local timestamping mechanism in order to avoid measurement errors due to network jitter.

The remainder of this paper is structured as follows. Section 2 presents the background of Software Defined Networks, the earlier works in monitoring SDN networks, and Available Bandwidth measurement in traditional networks. We present our method for measuring available bandwidth in SDN in Section 3. In Section 4 we address the main issues and limitations in SDN measurements. In Section 5 we discuss the main advantages offered by the newly proposed technique and possible applications that can benefit from them. In Section 6 we validate our ABW application using a Mininet based test configuration and also analyze the measurement error caused by the network delay. We propose an efficient and backward compatible extension to the OpenFlow protocol in Section 7 that adds message timestamping to further reduce measurement error. Finally, Section 8 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 [9]. In this paper we follow the definition of SDN as presented in [10], which is 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 devices. (ii) Forwarding rules are made based on a set of fields in the packet headers. This also guarantees unified behaviors 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.

Fig. 1 presents the architecture of Software Defined Networks. The SDN controller can communicate with the switch via the southbound API, where the most used standard is OpenFlow, and there are also other proposals, e.g. OVSDB [11], P4 [12] or ROFL [13]. For NOS platform there are many available open software such as NOX [14], POX [14], Floodlight [15] or Ryu [16]. Moreover, there are ongoing industrial consortia projects for controller platforms specialized for data centers, for e.g. OpenDayLight [17] or ONOS [18]. 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 SDN applications are only able to operate over one NOS platform. These northbound interfaces either uses a specific programing language (e.g. Java or Python) or a REST based API. We refer to [10] for a comprehensive taxonomy of different elements in Software Defined Networking.

2.1. Monitoring in SDN

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 [6] 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 [5] 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 then reroute 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.

Previous approaches do not rely on explicit time management in SDN, and on this specific topic we found that very little work has been published so far. One relevant work presents a variation on the Precise Time Protocol, named ReversePTP [19], aimed at distributing accurate time to SDN switches, allowing synchronized operations. An extension of the OpenFlow protocol has been proposed in [20] to add support for Synchronized Ethernet in SDN. Another approach, but focusing on delay, is presented in [21], providing bounds on the basis of the estimation of statistical traffic distribution. In such approach random sampling is performed on flow counters, in order to efficiently obtain the autocovariance of network flows; the autocovariance is then used to simulate the queue behavior of the switches and therefore numerically derive the bonds on queue length and packet delay. Our method provides estimates not based on statistical model estimation and subsequent simulation, even though we report statistical analysis aimed at evaluating the theoretical bounds for the estimation error. Besides the difference in the estimated performance metrics, and the use of statistical models and discrete-events simulation, in [21] the authors do not detail the error introduced by lack of time precision (possibly compensated for in the random sampling process).

The issues, goals and contributions considered in such works differ from ours and actually can be complementary to our proposal of introducing timestamping for OF messages.

2.2. Available bandwidth

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* [22].

For a formal definition, the available bandwidth is first defined on each link of a network path. For each time instant, the ith link is either inactive or trasmitting at its full capacity, so the average utilization of the link i in the time interval $(t-\tau, t)$ is

$$\bar{u}_i(t-\tau, t) \equiv \frac{1}{\tau}\int_{t-\tau}^{t} u_i(x)dx \tag{1}$$

and τ is the *averaging timescale*. The amount of traffic that is transferred over the link during the time interval $(t-\tau, t)$ is denoted as $l_i(t-\tau, t)$ and is equal to

$$l_i(t-\tau, t) = C_i \cdot \tau \cdot \bar{u}_i(t-\tau, t) \tag{2}$$

The *available bandwidth* in the time interval $(t-\tau, t)$ for the ith link, with capacity C_i, is

$$a_i(t-\tau, t) \equiv \frac{1}{\tau}\int_{t-\tau}^{t} C_i(1-u_i(x))dx \tag{3}$$

$$= C_i(1-\bar{u}_i(t-\tau, t))$$
$$= C_i - \frac{l_i(t-\tau, t)}{\tau} \tag{4}$$

In other words the available bandwidth of a link is the average of unused capacity during the considered time interval. The available bandwidth on a path is defined as the minimum value of available bandwidth of the links composing the path.

Available bandwidth 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.

2.3. Available bandwidth measurement methods

In traditional networks, available bandwidth estimation techniques are typically classified into active and passive (with the same definition provided in [23] for network measurement methods in general). Active techniques send probe packets into the network and analyze how network traversal affected their spacing/arrival to infer network status. Active ABW estimation techniques in the literature can be referred to two models, *probe gap* and *probe rate*, according to the hypotheses on the analyzed path and on the type of probing procedure adopted. Probe gap tools such as Spruce [24] or Traceband [25] 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 [26] and PathChirp [27].

Passive techniques for estimating the available bandwidth use multiple measurement points in the network to monitor bandwidth utilization, packet loss ratio, and packet delay. The available bandwidth can then be estimated if these measures are properly synchronized. These techniques are very complex to deploy thus they are rarely used in practice.

Table 1
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	ith 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 bandwidth on e_i, $a_i = c_i - b_i$
$P_{A \to B}$	The set of all available paths from A to B

A passive technique that leverages analysis methods developed initially for active ABW estimation is presented in [28], and consists in inspecting traffic traces generated by real applications running at the ends of the measured path, in order to detect the presence of packet trains similar to ones generated by active ABW estimation tools: for each of them the effect of network traversal is evaluated according to active estimation techniques, obtaining an estimate of the available bandwidth with no measurement overhead.

2.4. Main issues for traditional available bandwidth estimation techniques

The performance of most of active ABW estimation tools currently available is scenario-dependent and require non-trivial calibration [29,30]. The main issue they have in common is the limited accuracy, and systematic errors around 50% are not uncommon. Some of the tools (Diettopp and Pathload, the most accurate ones) have long convergence time, in the order of 10 s up to 40 s, and—depending on configuration settings and traffic conditions—they may not converge to an estimation. The approach using passive measurement with active-like analysis inherits the accuracy issues of the active techniques that are adopted in the processing phase, worsened by the impossibility of dynamically adjusting the characteristics of probing traffic (that is independently generated by the monitored applications). Estimation time is also dependent on the presence of suitable traffic generated by third party applications, therefore it is not predictable. These reasons lead the authors to propose it as a complementary method with respect to active tools.

3. Measuring available bandwidth in SDN networks

In SDN environments the situation is largely different from the traditional ones. The centralized control plane provides interesting opportunities for measuring the available bandwidth, which were unforeseeable in traditional environments. In the following we present our approach for the estimation of this important parameter and discuss the possibilities as well as the new challenges that SDN introduces in the ABW measurement field.

We propose the use of a passive technique for the Available Bandwidth estimation, 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 at any given time.

Using the northbound API of the NOS we query the topology abstraction of the network which is a mandatory feature in every SDN controller [10]. Firstly, our application uses this information to build up the 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 1). Due to the topology abstraction mechanisms the *capacity* c_i of every link is also known in the network.

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 *PortStatsReq* OpenFlow message. This method is already proven to be effective in SDN and it provides an easy solution for measuring the bandwidth utilization over the entire network. After this step, we calculate the *available bandwidth* a_i on every link in the network, using (4). Based on the a_i values we then calculate the available bandwidth on a given path P through the following equation

$$ABW_P = \min_{e_i \in P} a_i. \tag{5}$$

Our method is also able to distinguish between three different scenarios and calculate the ABW according to them. They are the following:

1. **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 (5). 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.
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 calculated through the following equation:

$$ABW_{A \to B} = \max_{P \in P_{A \to B}} \min_{e_i \in P} a_i. \tag{6}$$

   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 (5). 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).
3. **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 [31]. 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. Limitations and constraints for available bandwidth estimation in SDN

Based on our extensive analysis and measurements of the ABW over SDN networks, we have derived a number of limitations and constraints in estimating ABW with the technique we have proposed. For each of them we provide an analytical modeling of the issue, an experimental evaluation in emulated environment, and possible solutions or mitigations.

4.1. Measurement overhead

In traditional networks the measurement overhead caused by passive methods has been subject of several studies and proposals [4,6]. Due to both the architecture of SDN networks and the different possibilities for monitoring it provides, measurement overhead can have multiple aspects. We report in Fig. 3a visual breakdown of such aspects. Regarding *traffic*, measurement can affect the SDN control network (for passive methods), data plane network (for active methods), or both. As regards computation overhead, it can affect the logically centralized controller and the switches; for the switches the additional computations can impact the slow-path (whose primary duty is management, not monitoring), the fast-path (either directly or indirectly), or both.

In the case of the ABW estimation method we propose, the overhead in traffic regards the control network, and the computational overhead regards mainly the controller, as the switches are required a standard task (port counters readings). More specifically, due to periodic polling of switches counters, the control network is affected by additional traffic in the size of 80 bytes for every port (based on Section 7.3.5.5 in the OpenFlow 1.5.1 specification). This means that the statistics of 18 ports can be fitted into a single 1500 byte OpenFlow packet, thus in case of a 48 port switch a total number of 3 packets will be sent in every polling period. Even with a very frequent polling rate (e.g. polling the switches every second) this adds less than 5 Kbps traffic for every switch, which in a network with 200 devices makes a traffic overhead of less than 1 Mbps.

4.2. Accuracy limitation for lack of synchronization

In the simplest set-up with one physically centralized controller, the controller performs polling of measurement reports from switches at the parallelism level allowed from the networking infrastructure: if the controller and the switches are on a single flat control LAN, the controller has one single interface connected to such LAN, and the polling messages are sent as unicast messages to each of the switches, then requests are necessarily sequentially issued, possibly introducing a non-negligible delay among requests to switches. This delay is due to the relative ordering of the poll requests, and is additionally affected by a degree of randomness depending on the controller activities and LAN conditions. This delay adds to the transmission delays between the controller and each switch, whose impact is analyzed in Section 6.3. An upper bound estimation of error in the Available Bandwidth estimation due to lack of synchronization between switches polling times can be calculated noting that, said $\delta_{\max}$ the delay between the polling to the first switch and the polling to the last one in a single probing round, the maximum error on traffic load for links with capacity C is

$$\delta L_1 = C \cdot \delta_{max}$$

and on two subsequent pollings, in the worst case the error on throughput estimation is

$$B_{err} = \frac{\delta L_2 + \delta L_1}{\tau} = \frac{2 \cdot C \cdot \delta_{max}}{\tau}$$

where τ is the polling rate. The maximum delay in polling $\delta_{\max}$ in turn depends on the number of switches to be polled N, the capacity of the link connecting the controller to the control network C_{ctl} and the length of the query packet Len_{query}:

$$\delta_{max} = (N-1)\frac{Len_{query}}{C_{ctl}}$$

in the hypothesis that all messages are put on the wire back-to-back with no additional delay (e.g. due to context-switch). With the simplifying assumption that all links have the same capacity $C = C_{ctl}$, we obtain

$$B_{err} = \frac{2 \cdot (N-1) \cdot Len_{query}}{\tau}$$

Given the size of *FlowStatsReq* message $Len_{query} = 56$ Bytes,[1] a controller managing $N = 101$ switches, with poll rate $\tau = 0.5$ s can suffer up to 44.7 Kbps error on throughput estimation, that on a 1Gbps link is a $4.4 \cdot 10^{-5}$ relative error on flow rate estimation. Even in this case, the higher the polling rate, the higher the error.

[1] OpenFlow Switch Specification Version 1.5.1 Section 7.3.5.2

4.3. Critical time-scale dependence of estimation

For its very definition, ABW is a metric that depends on a time interval (see Eq. (4)), thus its dependence on the choice of the *averaging timescale* τ is self evident. Traditionally such parameter has been set according to monitoring needs, using time intervals usually in the order of the minutes, down to 30 s [32], while techniques provides a snapshot of the status of the whole path under measurement, averaging on sub-second time scale: application traffic can take more than this time to traverse the path, while network conditions change in the meanwhile. In general, polling delay should be chosen so that traffic entering the controlled SDN network can exit it within a single ABW estimation period. Said $\delta_{ingr-egr}$ the maximum delay for a packet to traverse the SDN network, and τ the polling period, this condition translates in $\tau \gg \delta_{ingr-egr}$. While in a fully wired setup this is not a strict constraint, if the network includes wireless links these can significant add to the border-to-border delay thus this constraint has to be accounted for. If this is not the case, traffic could traverse the network while the controller changes its internal representation of network status, and not even a single packet would experience the path available bandwidth as estimated by the controller. According to the usage of ABW estimation, failing to enforce this constraint can lead e.g. to the invalidation of routing decisions on the run, or erroneous granting or denying access to flows, if access control is applied. Therefore the practical applicability of ABW estimation with fine granularity in time is to be checked against both polling time and border-to-border transmission time.

4.4. Accuracy limitation for lack of timestamp

Due to the lack of a switch-generated timestamp in the OpenFlow message, the instant when the reading is performed is unknown, and has to be estimated by the receiving controller; this introduces uncertainty associated with the processing and transmission delay between the switch and the controller.

In Fig. 4 we report a message sequence chart depicting the communications between the ABW measuring application, the controller and the switches, during the measurement process; in the chart we name the time instants associated with notable events. With reference to Fig. 4, as no network delay is implied, the difference between *t-poll 1* and *t-req 1* is in the order of *μs*, as well as the difference between *t-read 1* and *t-meas 1*; the same applies to the each polling. We will therefore approximate *t-poll 1* with *t-req 1, t-read 1* with *t-meas 1* and focus on the time laps between *t-gt 1*, when the counter values are read by the switch and the ground truth timestamp is extracted, and *t-meas 1*, when the ABW application time-stamped the received the message. By defining as ΔL the difference between the counter values L_1 and L_2 in the first and second measurements, respectively, we obtain the following formula for the error:

$$\epsilon = \frac{\frac{\Delta L}{T^2_{GT}-T^1_{GT}} - \frac{\Delta L}{T^2_{meas}-T^1_{meas}}}{\frac{\Delta L}{T^2_{GT}-T^1_{GT}}} = \frac{\frac{1}{T^2_{GT}-T^1_{GT}} - \frac{1}{T^2_{meas}-T^1_{meas}}}{\frac{1}{T^2_{meas}-T^1_{meas}}} \tag{7}$$

In case we assume that the counter value extractions happen at perfect τ rate ($\forall i : T^{i+1}_{GT} - T^i_{GT} = \tau$), and mark δ_i as the network delay between the application and the ground truth timestamping in the ith measurement period ($\delta_i = T^i_{meas} - T^i_{GT}$), Eq. (7) can be simplified as:

$$\epsilon = \frac{\tau}{\tau + \delta_2 - \delta_1} - 1 = -\frac{\delta_2 - \delta_1}{\tau + \delta_2 - \delta_1} \tag{8}$$

The formula in Eq. (8) suggests that the error due to the timestamping mechanism is not dependent on the network delay itself but rather the difference between the delay in consecutive measurement intervals, hence a jitter like metric. For example, if we model the network delay as $\delta = \delta_{\min} + x$, where $\delta_{\min}$ is the minimal possible delay between the switch and the ABW application and x is a positive random variable, $\delta_{\min}$ will fall out from the equation. This means that the network distance between a switch and the ABW application will not affect the accuracy of the measurements as long as the jitter in the network is under control. Furthermore, the magnitude of the error will be in the range of the ratio between the network jitter and the polling period.

To confirm the above assumption we introduce a simple case where the network delay has normal distribution and analytically calculate the error. Based on Eq. (8) the distribution of the error can be calculated as the difference of two independently and identically distributed normal variable. Thus if the distribution of the delay is $\mathcal{N}(m, \sigma^2)$, the distribution of the difference of two consecutive values will be $\mathcal{N}(0, 2\sigma^2)$, thus the distribution of the error will be $\mathcal{N}(0, \frac{2\sigma^2}{\tau})$, if $\tau \gg \sigma$. Based on this formula we can calculate the mean of the absolute error by the following.

$$\begin{aligned} E(|\epsilon|) &= \int_{-\infty}^{\infty} \left|\frac{y}{\tau - y}\right| f_y(y)dy \\ &= \int_0^{\infty} \frac{y}{\tau - y} f_y(y)dy + \int_0^{\infty} \frac{y}{\tau + y} f_y(y)dy \\ &= \int_0^{\infty} \frac{2\tau y}{\tau^2 - y^2} f_y(y)dy \\ &\overset{\tau \gg \sigma}{=} \int_0^{\infty} \frac{2y}{\tau} f_y(y)dy = \sqrt{\frac{4}{\pi}}\frac{\sigma}{\tau} \end{aligned} \tag{9}$$

Eq. (9) describes that if the distribution of the network delay is normal, than the mean measurement error will be a linear function of the ratio between the standard deviation of the delay and polling period. Further, it also tells that the mean error will be independent on the mean of the network delay. Such scenarios will be emulated and the results will be evaluated in Section 6.3.[2]

We explicitly notice that, even if affected by said sources of error, ABW estimations provided by our technique is orders of magnitude better than the ones provided by active tools in traditional networks. Moreover, the time-scale dependence is meaningful in our case just because high frequency estimation has become possible, while is unfeasible in traditional networks. As a counterpart of said limitations and caveats, our technique presents several advantages and enables new applications of ABW, not previously considered in the traditional scenario: we discuss such advantages and enabled applications in the following section.

5. Advantages and novel applications of available bandwidth estimation in SDN

The newly presented technique for estimating ABW offers several advantages over the active techniques in traditional scenarios. First, it is conceptually much simpler, not needing assumptions on the statistical properties of cross traffic, neither analytical models of the devices composing the paths under measurements. This has an impact on both the accuracy of the estimation, on the applicability of the technique, and on the lack of necessity of a per-scenario tuning. Second, it allows a for a time granularity and estimation accuracy of orders of magnitude better than the ones reached by active techniques in traditional scenarios (see Section 6.3 for more details), making several unprecedented applications possible.

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

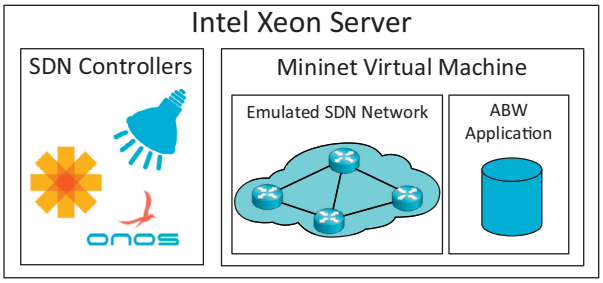


Fig. 2. The assembled test configuration.

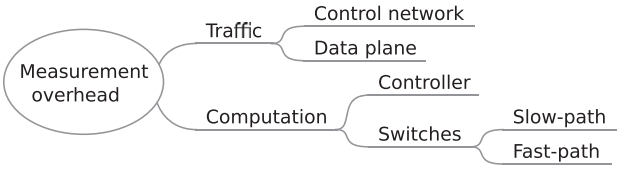


Fig. 3. The impact of measurement in SDN networks.

Based on such unprecedented characteristics in terms of accuracy and high frequency of estimation, we believe that several novel applications based on ABW knowledge provided by our method are possible:

- Highly-dynamic routing [33,34].
- No-resv admission control – instead of checking availability of resources per-request like a RSVP, controller already has the knowledge to admit/refuse a new flow based on ABW [35].
- Traffic consolidation – like in NFV, where processing is consolidated on busy servers, to shut down unused ones and save costs, the same can be done for network (virtualized) devices and links: as long as there is available bandwidth, traffic can be routed so as to minimize the number of links and devices needed. This also results in the reduction of probing/control overhead, as sleeping switches do not cause/require control [36].
- Adaptive video – as today's killer application is video streaming, and DASH is expected to be the future standard for adaptive video transfer. A video player could ask for the current ABW conditions from the controller to set up the best available resolution which will not congest the network (instead of using very poor ABW estimation), or the controller could manage DASH traffic and its competing traffic according to the available bandwidth and monitored QoE [37].

6. Experiments over mininet testbed

In order to evaluate our available bandwidth measurement application presented in Section 3 and also, to validate our statements in Section 4, we conducted extensive network emulation scenarios using Mininet [38]. Fig. 2 presents the schematics of our testbed. During the emulation scenarios SDN switches are represented as running Open vSwitch entries and they can be connected to three different SDN controllers. Mininet emulation runs the switches inside a separate virtual machine[3] on the host server using VirtualBox. Virtualizing the SDN environment is the suggested approach by the developers of Mininet since it makes research results easier to reproduce and build upon [39]. However, we chose to run the controllers directly in the host server since we often faced load issues when we run it inside the virtual machine.

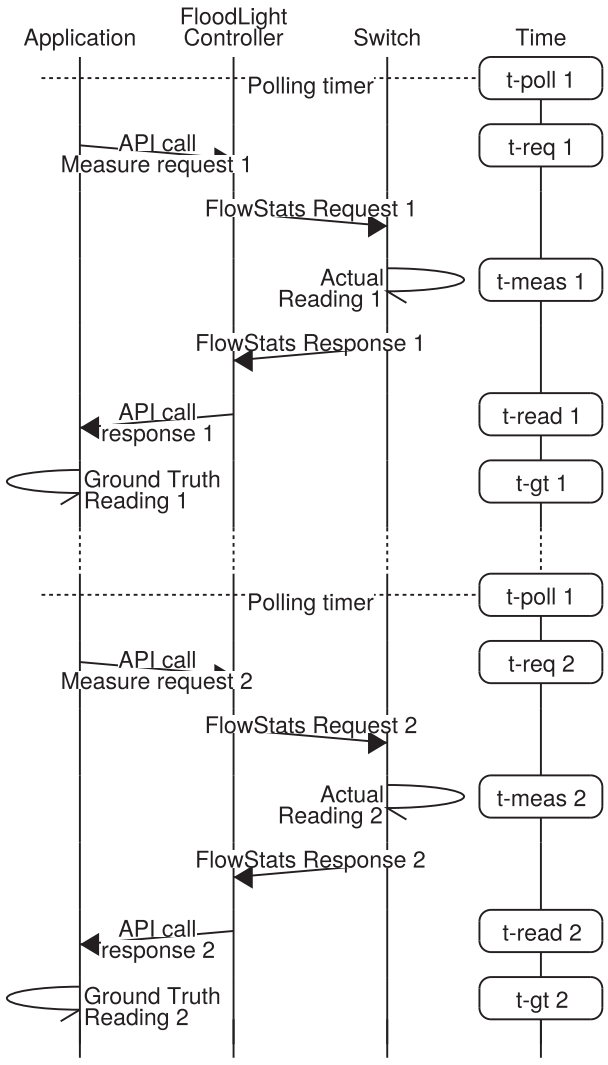


Fig. 4. The measurement process.

Table 2
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	Ubuntu 14.04 64-bit
VM configuration	4 CPU cores, 2 GB memory
Mininet version	2.2.0
OVS version	2.0.2
Floodlight version	1.0
OpenDaylight version	0.4.3 Beryllium SR3
ONOS version	1.6.0 Goldeneye

For the proof-of-concept application in Section 6.2 and 6.3 we used Floodlight [15] as NOS since it is easy to use and provides the exact features we needed to validate the theoretical results.[4] In Section 6.4 we created the same application for industrial controller platforms OpenDaylight [17] and ONOS [18] and we found fundamental differences on how they collect the statistics from the SDN switches. For further reference, we collected the used hardware and software versions in Table 2.

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

[4] Preliminary results in the same framework of this paper have been published in [40]

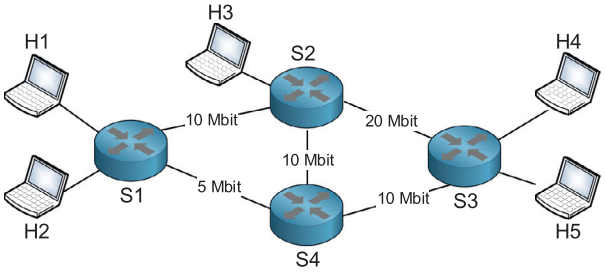


Fig. 5. The test topology in Mininet.

During the emulation scenarios concurrently with the measurement application, we used a kernel polling mechanism for generating reference values for the measurements. Mininet creates separate virtual Ethernet interfaces in the Linux system for every interface of the emulated SDN switches. We use IPtables to obtain reference measures regarding the traffic on the interfaces. Packets between the ABW application and the SDN controllers can (and in some of our test cases will intentionally) suffer variable delays.

6.1. Test configuration

Fig. 5 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 [29]. 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 tight link will become different, which is an important test case for the calculations of current ABW tools [29]. To realize such scenario, we use *TrafficControl* to control the capacities of the links and also, (in some scenarios) to add variable delay between the polling application and the Floodlight controller.

We avoided sending any traffic through S4, as the default route policy in all the three controllers do. This was 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.

6.2. Validation of available bandwidth application

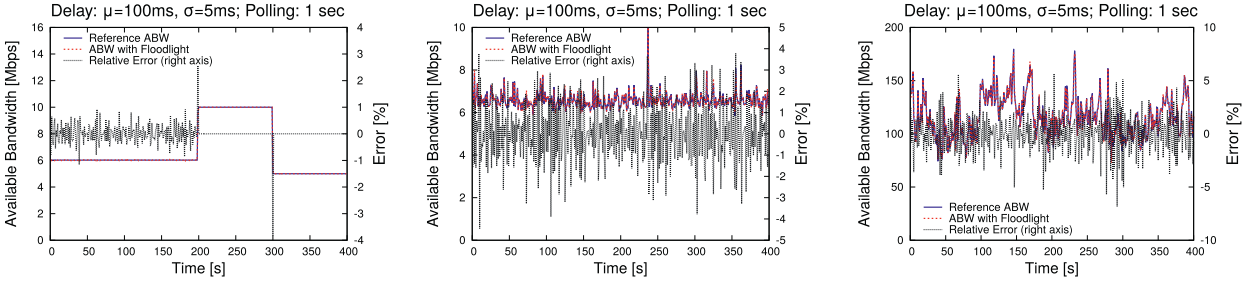
During the validation process we use D-ITG [41,42] for traffic generation, since it was proven to work much reliably than other traffic generation platforms [43]. Using D-ITG, we defined the following three traffic scenarios in order validate the our ABW application in different circumstances:

1. *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 s, 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. 6 a shows the ABW on the best route from H1 to H5 (the path offering the maximum ABW). Reported values are the estimated ABW, the reference ABW (as derived from packet capture), and the relative error. Although the bandwidth measured by the Floodlight controller is varying due to the variable latency introduced between the controller and switches, 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 route is S1→ S4→ S3 with 5 Mbps available bandwidth (and no traffic). After one measurement period (1 s long, at second 200 and second 300) affected by transitory error, the estimated ABW stabilizes to the (constant) actual ABW, with no error.
2. *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. 6b plots the ABW of the best path from H1 to H5 (estimated by Floodlight, reference value, and relative error). In this case we used $\lambda = 1.75$ as shape parameter for both flows, whereas for scale parameter we used $X_{H1} = 1$ ms and $X_{H3} = 0.5$ ms 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.
3. *Real traffic.* We collected real traffic measurements from the campus network of the Budapest University of Technology and Economics. We used a Cisco 6500 Layer-3 switch that aggregates the traffic of two buildings and linked them to the core layer of the network. A 10-minute-long trace was used for this purpose. The trace was recorded in No. 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. 6c presents 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.

As it can be observed on the left axis the three plots in Fig. 6, in every scenario there is some slight deviation between the implemented REST API polling based measurements and the reference values. This is due to fact that the timestamp value is created at the ABW application rater than being provided by the switches when reading the counters (i.e. taking the measurements). The monitoring packets suffer variable delay on the network, which causes the error that we described in Section 4.4.

6.3. Analysis of measurement error

In order to fully understand the measurement error caused by the lack of local timestamping at the SDN switches, we conducted several Mininet emulation scenarios using a wide range of network delay and polling rate setups. Firstly, we set up no delay extra delay on the which can be considered as an ideal case. Table 3 reports the mean and the standard deviation of the measured error values for such cases. 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. Note that we always use the polling period in the denominator during the error calculation in Eq. (8) thus a more frequent polling generates more reference values but with a slightly larger error. One could average the values from a more frequent measurement in order to get a more precise on a larger timescale but we found that the er-



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

Fig. 6. Measuring bandwidth on the link between S2 and S3 and the available bandwidth over the Mininet test network.

Table 3
Error rate of ABW measurements using real traffic replayed by D-ITG with no added delay.

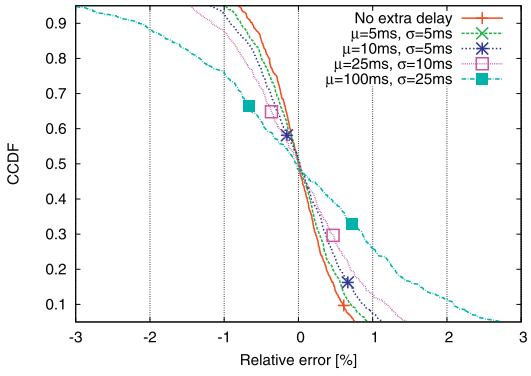
		Polling period in seconds								
		0.5	1	2	5	10	20	30	40	60
Error	Mean	$7.2 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	$1.8 \cdot 10^{-3}$	$8.5 \cdot 10^{-4}$	$5.5 \cdot 10^{-4}$	$2.1 \cdot 10^{-4}$	$1.3 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$6.5 \cdot 10^{-5}$
	STD	$9.4 \cdot 10^{-3}$	$5.1 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$	$7.4 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$	$1.6 \cdot 10^{-4}$	$1.3 \cdot 10^{-4}$	$6.9 \cdot 10^{-5}$

ror is similar to using a larger polling period which generates less overhead. If we consider the minute time scale, where typically the current ABW tools operate [30], the average error rate is less than 10^{-5} which can be considered as very accurate.

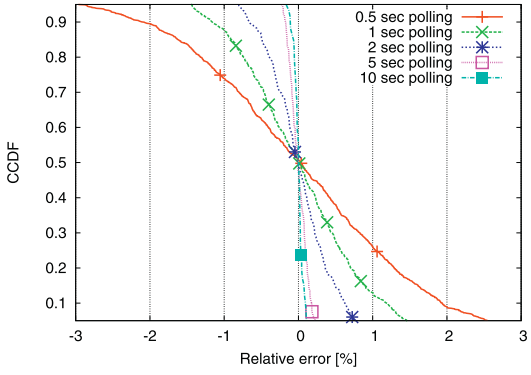
Hereafter, we present the cases were we introduced artificial delay between the SDN switches and the Floodlight controller to investigate its effect over the error rate. Fig. 7 presents the CCDF of the error for different delay values using frequent polling rates. In Fig. 7a we fixed the polling period to 1 sec and used different delay values between the switches and the Floodlight controller. In details, we added delay values 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.

In Fig. 7b 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 s and 10 s. The results confirm our previous calculation in that increasing the polling period the measurement error decreases linearly. This phenomenon can also be justified as 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 network jitter 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.

In Fig. 8 we plotted the result of every measurement with real traffic. We used fifteen different mean-std delay setups for all the five polling periods and emulated every scenario four different times. The measured mean delays in all the 300 measurement cases are depicted against the mean and the standard deviation of the delay in Fig. 8a and b, respectively. The results in Fig. 8a confirms our statement in Section 4.4 and shows that there is no correlation between the mean error of the ABW measurement and the mean of the delay introduced to the system. However, Fig. 8b is a clear indication that there is a linear dependence between the standard deviation of the monitoring packet delay and the mean error of the available bandwidth measurement which confirms the calculations in Eq. (5).



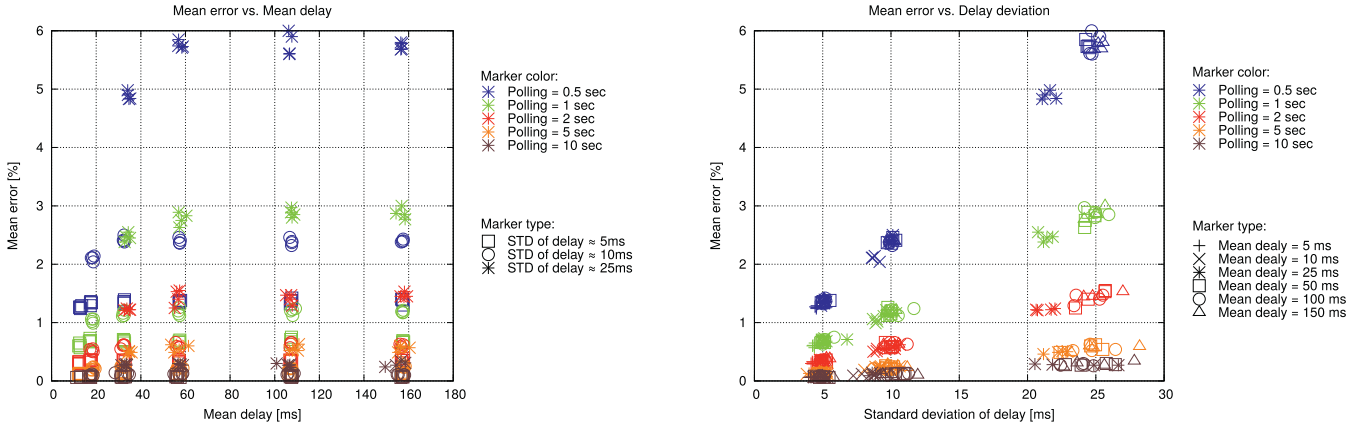
(a) Error values for 1 ms polling using different delay.



(b) Error values for 20ms delay using different polling periods.

Fig. 7. The CCDF of the relative errors of the ABW measurements using the Floodlight controller compared to reference values (range is limited to [0.05, 0.95] quantiles).

Table 4 summarizes the mean and the standard deviation of the measurement result in the most interesting cases. These results 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.



(a) Average error measured with different mean delay setups.

(b) Average error measured with different delay variation setups.

Fig. 8. The average error measured during real traffic replay using different mean delay and standard deviation setups.

Table 4
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%
	$\mu = 5$ ms $\sigma = 5$ ms	1.29%	1.62%	0.6%	0.75%	0.3%	0.37%	0.13%	0.16%	0.06%	0.07%
	$\mu = 10$ ms $\sigma = 5$ ms	1.34%	1.67%	0.73%	0.93%	0.36%	0.45%	0.19%	0.28%	0.07%	0.09%
	$\mu = 25$ ms $\sigma = 10$ ms	2.5%	3.11%	1.12%	1.43%	0.62%	0.78%	0.24%	0.29%	0.14%	0.16%
	$\mu = 100$ ms $\sigma = 25$ ms	5.6%	7.04%	2.28%	3.59%	1.47%	1.85%	0.54%	0.66%	0.3%	0.37%

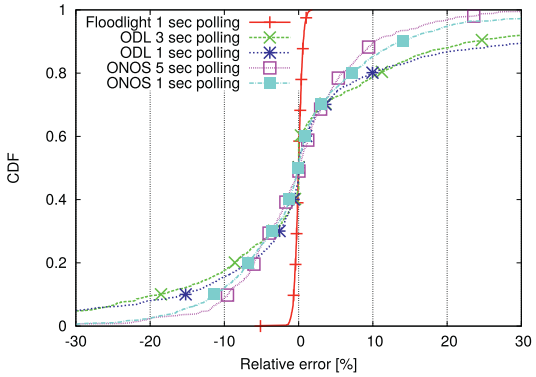
6.4. Measurements with industrial controller platforms

In order to test out application in a wider spectrum we also prepared the same ABW application over OpenDaylight (ODL) [17] and ONOS [18], that are among the most commonly used SDN controllers in industrial environment. Since both of these controllers provide an API access for querying the switch port counters we only had to modify the API URL in the request and the JSON parser module for processing the replay. However, the APIs of both ODL and ONOS work very differently than what we assumed for our ABW application and what we presented in Fig. 4. When we initiate an API request for the port counters these controllers do not send an OpenFlow *PortsStatsRequest* message to the SDN switch but rather sends back the latest measured value. The port counter values are collected by a separate statistics collector module which has to be installed via the controllers CLI. These modules have a default polling period (3 sec in ODL and 5 sec in ONOS), but there is the possibility to modify this rate. However, both ODL and ONOS do not provide a timestamp value for the measured port counter values when we query them via the APIs. This phenomenon made our application to estimate the available bandwidth very poorly since our measurement (with the reference value generation) was out of phase from the collection of the controllers. Fig. 9a present the CDF of the measured error rate with the two controller in two setups, i) using the default polling rates (3 sec in ODL and 5 sec in ONOS), and ii) using 1 sec polling rate in both our application and end the controllers' statistics collector module. We also conducted measurement when the polling rate in our application was not the multiple of the polling rate in the controller, but those result were completely missing the real values on the network. Based on these result we suggest to extend the API and the statistics collector module of both ODL and ONOS with a timestamping mechanism in order to have a reference when the given values were measured. We expect that using such timestamping would significantly decrease the error for an ABW application over both ODL and ONOS.

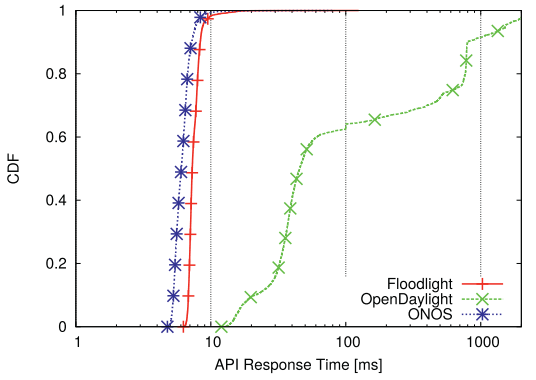
Another interesting result of these measurements is that the rest API of OpenDaylight has a very slow response time, in fact on average we measured almost two orders of magnitude larger response times than in case of Floodlight and ONOS. We reported the CDF of the API response times for the three considered controllers in Fig. 9b. Since ODL is a very complex software platform we couldn't find the reason for this problem but several forum entries suggested that other technicians also experienced similar slowness. The average response time of ONOS and Floodlight is around 5 ms and 7 ms, respectively. The slightly slower response in Floodlight compared to ONOS is expected since Floodlight sends an OpenFlow *PortsStatsRequest* message after every API request and only sends back the response after getting the counter values from the SDN switch.

7. Novel features for OpenFlow

While the access to switches counters highly simplifies most monitoring tasks, we have theoretically (Section 4.4) and experimentally (Section 6.3) verified that the lack of a timestamp in OpenFlow (OF) messages introduces a source of uncertainty and thus a measurement error for the estimation of the traffic rate and available bandwidth. The introduction of a switch-based timestamp would further reduce or completely remove the source of



(a) Error value comparison of different controllers.



(b) API response time comparison of different controllers.

Fig. 9. The CDF of ABW measurement error and API response time generated by our application over OpenDaylight, ONOS and Floodlight.

uncertainty in the estimation of ABW, therefore we advocate for an extension to the OpenFlow standard, adding timestamping of OF messages, and propose requirements and different implementation possibilities for it in the following.

The timestamp should be generated as close as possible in time to the counter readings. Moreover the sequence *(timestamping, reading)* should be atomic with regards to packet processing, i.e. no intervening packet should be accounted for until all the counters are read, in order to provide a consistent snapshot in time of flow statistics. If those two conditions are met, and in addition the clock resolution for timestamping does not introduce further uncertainty (e.g. 1 µs resolution will allow for 125 bytes counter resolution on 1 Gbps links), and between the timestamp and the reading of all counters no packets are completely received, the error on ABW estimation will be zero.

For the implementation of the timestamp extension, a specific request and reply format has to be defined, and the reply has to carry a representation of a time instant. The timestamp could be represented as a 64 bit structure such as POSIX `timespec` specification ,[5] representing UNIX *epoch* time (seconds from January 1st 1970) reserving 32 bits for seconds (truncated to integer) and 32 bits for remaining nanoseconds (approximated to the nearest integer, with resolution dependent on the implementation). This format is the same adopted for the standard Network Time Protocol (NTP) timestamps [44]. For the message formats we propose different implementations, described hereafter.

7.1. Experimenter-based implementation.

Possible implementations of timestamp extension on flow statistics can leverage the "experimenter" messages ("vendor" messages in OFv1.0) to define same format and function of a *FlowStats* reply with a timestamp field added. This implementation would provide possibly the benefit of reusing all the code of *FlowStats* message generation (in the switch) and parsing (in the controller), and not modifying the OF standard, being compatible with OF versions since 1 on. The protocol overhead for the reply message would be, besides the aforementioned 64 bit timestamp, an `experimenter_header` adding two 32 bit fields `experimenter` and `exp_type` as defined by the "OpenFlow Switch Specification Version 1.5.1 Section7.5.5" and valid for OFv1.1 up to OFv1.5.1 (in OFv1.0 only 32 bits are required for a single `vendor_id` field). For the request message the protocol overhead would be just the 64 bits of the `experimenter_header` in addition to the standard OF header (see Fig. 10 for the format of the proposed implementation for the generic Experimenter message with the addition of a Timestamp).

7.2. Protocol-wide modification.

The implementation proposed in the previous section for the *FlowStats* messages can be applied also to other messages, by defining an *experimenter* message type for each of the timestamp-augmented version of the standard OF message. Wrapper code, with reuse of the related non-timestamped OF message for construction (in the switch) and parsing (in the controller), would be implied for each of the considered messages. Moreover, an high-precision timestamp on all switch messages can be useful for multiple uses, such as performance evaluation, troubleshooting, and security. These considerations suggest to include the timestamp in the OF header, common to all OF messages. On the other hand, both the implementation and computational cost of the timestamping operations, and the additional space required in the messages generated by the switch, suggest to have the timestamp as an optional field. Moreover, a variation in the OF header is a major change in the protocol, thus compatibility issues have to be carefully accounted for. We propose an implementation that introduces the optional request for timestamped reply in the OF protocol, while retaining backward compatibility for the header format, by means of a change in the definition of the OF header field `type`, that is common to all OF messages. In OFv1.5.1 `type` is allocated 1 byte in the header, and values from 0 to 35 are assigned to OF messages .[6] We propose to reserve the most significant bit of `type` to manage the timestamping option, as a *Timestamp Flag*, leaving 128 possible values for OF message types. Setting to 1 the flag will signify "Timestamp Required" for messages carrying requests, and "Timestamp Provided" for replies; when such flag is set to 0 the format, meaning, and associated functions of already defined OF messages remain the same as per OFv1.5.1, retaining full backward compatibility.

In request messages with *Timestamp Flag* set there is no protocol overhead; in reply messages with *Timestamp Flag* set, additional 64 bits will be appended to the OF header, changing the total length of the header, and containing the timestamp as defined previously (Fig. 11). Parsing a "Timestamp Required" request a switch will mask the *Timestamp Flag* as 0 and process the message as a standard OF message, generate the timestamp, and append it to the reply header, that will have the "Timestamp Provided" flag set. This will avoid message-specific wrapper definitions, and for standard (non-timestamped) OF messages will imply

[5] http://pubs.opengroup.org/onlinepubs/007908799/xsh/time.h.html

[6] OpenFlow Switch Specification Version 1.5.1 Section 7.1.1

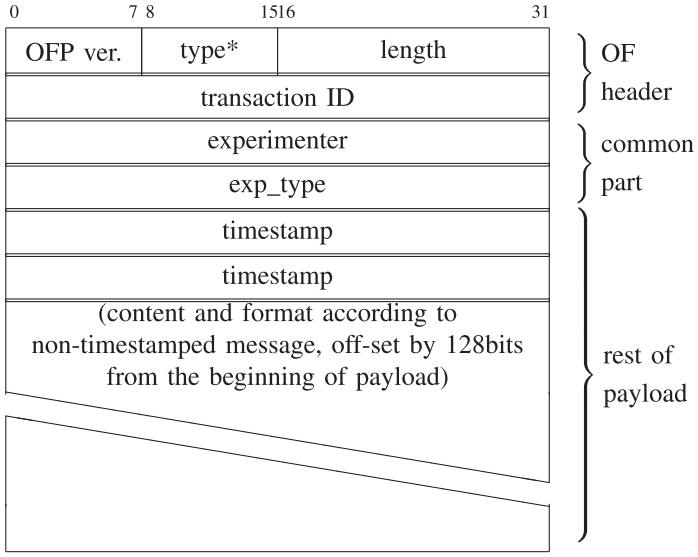


Fig. 10. Format of OpenFlow Experimenter message with Timestamp. * *type* header field is set to value `OFPT_EXPERIMENTER`.

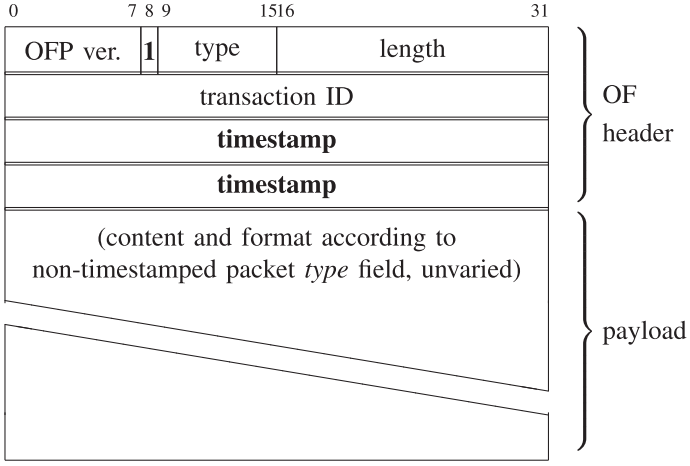


Fig. 11. Format of OpenFlow reply packet with *Timestamp Flag* set. Changes against OFv1.5.1 are highlighted in bold.

just a bit check in addition to the current processing flow. Analogous mechanism is adopted to parse a "Timestamp Provided" reply by a controller: the *Timestamp Flag* is checked, and if set the additional timestamp field is extracted from the header and stored, then the *Timestamp Flag* is masked to 0 and the body part of the message is processed according to the standard (non-timestamped) OF message type. Also in this case, for standard OF messages the only additional processing is a single bit check.

7.3. Discussion

An extension similar to the one we described has been presented in [45] in a more general framework, with the aim of enforcing synchronization of OF configuration updates. Besides the different goal, the structure proposed in [45] differs from ours as it includes a Type-Length-Value (TLV) generic experimenter header, that would precede at least one time TLV, which in turn adds more fields to time values alone (namely: type, length, flags, and padding). Moreover the time representation chosen in [45] uses 64 bits for seconds, while we adopted 32 bits. As the Time-TLV extension format is intended both for requests and reply, even if the request does not carry a time stamp (that would be useless for our purposes anyway), an overhead is present for both messages: for the request message the difference is of 256 bits and for the reply 192bits leading to a total of 448 bits (300% additional overhead on the original message) with respect to our experimenter-based proposal, and 704 bits in addition to our bit-based extension (1,100% additional overhead), while offering no additional features for what concerns our aims.

The *Scheduled Bundle* OF extension, adopted in the OF standard version 1.5, can be used to enforce the atomic execution at a specified time of a group of commands. In our case we could in principle exploit it to let the Controller enforce the precise polling time on the switch. By enforcing the reading of counters at a given time, the errors due to timing as analyzed in Section 4.2 and 4.4 would be significantly reduced (at least in principle, still depending on synchronization accuracy and on precision in meeting the deadline). However, due to the original intended usage for the extension, there is significant overhead compared with our proposed extension, both in term of messages (minimum 3 to require the reading), time (at least one additional RTT) and of data (1,100% the overhead over the standard OF message that is timestamped, with respect to our proposed implementation). Being this reiterated for every poll request, for every switch, we think that such overhead should be taken into account, and the implementation using Scheduled Bundles should be considered unfit for our purposes. Other minor aspects add to this, e.g. synchronization is required among all involved switches and the Controller. In our case if such constraint is relaxed, our proposed Timestamp extension would be affected by the error considered in Section 4.2, but still would allow to remove the error discussed in Section 4.4.

Given the different layers of SDN, ABW estimation can be implemented as an SDN application or as SDN metric, according to the support from the controller and the switch, using in order of decreasing accuracy:

1. protocol-wide timestamping (OF protocol modification: change in standard definition, switch, and controller)
2. experimenter-based timestamping (Experimenter messages definition: change in switch and controller)
3. controller-based timestamping (controller extension: change in controller only)
4. application-based timestamping (implemented as SDN application)

If the controller supports the implementations from 1 to 3, the best method can be adopted on a per-switch basis thanks to the negotiation phase, when the switch can advertise supported capabilities.

8. Conclusion

In this paper we tackled the problem of Available Bandwidth estimation and monitoring in Software Defined Networks. We presented the state-of-the-art techniques for this emerging architecture, emphasizing how they utilize the new features available and what are their limitations. We also analyzed the source of errors introduced by SDN and openflow, analytically deriving the measurement error due to lack of local timestamping mechanism in OpenFlow. The analytical results have also been validated on a testbed implemented in Mininet using different kinds of traffic, also comprising real traffic traces collected on a real network. 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 jitter.

We also implemented our ABW application over the two most commonly used industrial SDN controllers: OpenDaylight and ONOS. We showed that these controllers do not provide the necessary features in order to accurately measure the available bandwidth. Due to the different mechanism in ODL and ONOS the reported measurement errors were more than one order of magnitude worse compared to our proof of concept implementation over Floodlight. Given these results we suggested to extend the API and the statistics collector modules of both ODL and ONOS with a timestamp providing mechanism.

We finally proposed an extension to the OpenFlow protocol providing local timestamping mechanism in order to avoid measurement errors due to network jitter. In particular, we proposed two implementations of such feature. One implementation leverages the “experimenter” message type, provided by the OF standard purposely for extending the base OF capabilities. Due to the general potential of the proposed extension, and the inefficient implementation through “experimenter” messages, we also propose an amendment to the OF standard by re-purposing one bit of the current protocol as a flag to signal the request or presence of a timestamp. By reducing the maximum number of different OF message types from 256 to 128 (up to 35 are used in OFv1.5.1) we can efficiently extend the capability of the OF protocol in a fully backward-compatible way and no additional protocol overhead.

In our ongoing work we are investigating the possible impact of the specific implementation of the NOS and the proposed implementations of timestamp capability on the accuracy and timeliness of other measurement activities, as well as applications of our technique to hybrid scenarios mixing SDN and traditional networks. We are also considering expanding the possible implementations of ABW estimation based on meter statistics (leveraging `duration` fields in meter stats introduced in OFv1.3), and the use of approaches based on In-Network-Telemetry [46].

Acknowledgments

This work is partially funded by the Ministry of Research of Italy (MIUR) under the Art. 11 DM 593/2000 for NM2 srl (Italy) and, for the University of Napoli, by the SOMETIME project, part of the MONROE EU Project funded by the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 64439.

P. Megyesi and S. Molnár are working in the High Speed Networks Laboratory which is supported by Ericsson.

The authors also want to thank the anonymous reviewers for their insightful and helpful comments on an earlier version of this paper.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[2] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H. Madhyastha, Flowsense: Monitoring Network Utilization with Zero Measurement Cost, in: Passive and Active Measurement, in: Lecture Notes in Computer Science, 7799, 2013, pp. 31–41.
[3] M. Jarschel, T. Zinner, T. Hohn, P. Tran-Gia, On the accuracy of leveraging sdn for passive network measurements, in: Australasian Telecommunication Networks and Applications Conference 2013 (ATNAC '13), 2013, pp. 41–46.
[4] S. Chowdhury, M. Bari, R. Ahmed, R. Boutaba, Payless: a low cost network monitoring framework for software defined networks, in: Network Operations and Management Symposium (NOMS), 2014, pp. 1–9.
[5] N. van Adrichem, C. Doerr, F. Kuipers, Opennetmon: Network monitoring in openflow software-defined networks, in: Network Operations and Management Symposium (NOMS), 2014 IEEE, 2014, pp. 1–8.
[6] D. Raumer, L. Schwaighofer, G. Carle, Monsamp: a distributed sdn application for qos monitoring, in: Federated Conference on Computer Science and Information Systems (FedCSIS), 2014.
[7] K. Phemius, 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, 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] N. Feamster, J. Rexford, E. Zegura, The road to sdn: an intellectual history of programmable networks, SIGCOMM Comput. Commun. Rev. 44 (2) (2014) 87–98.
[10] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: a comprehensive survey, Proc. IEEE 103 (1) (2015) 14–76.
[11] B. Pfaff, B. Davie, The Open vSwitch Database Management Protocol, RFC, 7047, Internet Engineering Task Force, 2013.
[12] 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, in: HotSDN '13, 2013, pp. 127–132.
[13] M. Sune, V. Alvarez, T. Jungel, U. Toseef, K. Pentikousis, An openflow implementation for network processors, in: Third European Workshop on Software Defined Networks (EWSDN), 2014, pp. 123–124.
[14] NOX and POX SDN Controllers, [Online]. Available: http://www.noxrepo.org/.
[15] Floodlight, Available: http://www.projectfloodlight.org/.
[16] RYU Network Operating System, [Online]. Available: http://osrg.github.com/ryu/.
[17] The OpenDayLight Project, Available: http://www.opendaylight.org.
[18] The Open Network Operating System (ONOS), [Online]. Available:http://onosproject.org/.
[19] T. Mizrahi, Y. Moses, Using reverseptp to distribute time in software defined networks, in: Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), 2014 IEEE International Symposium on, IEEE, 2014, pp. 112–117.
[20] R. Surez, D. Rincn, S. Sallent, Extending openflow for sdn-enabled synchronous ethernet networks, in: Network Softwarization (NetSoft), 2015 1st IEEE Conference on, 2015, pp. 1–6, doi:10.1109/NETSOFT.2015.7116183.
[21] Z. Bozakov, A. Rizk, D. Bhat, M. Zink, Measurement-based flow characterization in centrally controlled networks (2016).
[22] R. Prasad, C. Dovrolis, M. Murray, K. Claffy, Bandwidth estimation: metrics, measurement techniques, and tools, Netw., IEEE 17 (6) (2003) 27–35.
[23] A. Morton, Active and Passive Metrics and Methods (with Hybrid Types In-Between), RFC, 7799, Internet Engineering Task Force, 2016.
[24] J. Strauss, D. Katabi, F. Kaashoek, A measurement study of available bandwidth estimation tools, in: Proceedings of ACM SIGCOMM Conference on Internet Measurements, 2003, pp. 39–44.
[25] C.D. Guerrero, M.A. Labrador, Traceband: a fast, low overhead and accurate tool for available bandwidth estimation and monitoring, Comput. Netw. 54 (6) (2010) 977–990.
[26] M. Jain, C. Dovrolis, End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput, IEEE/ACM Trans. Netw. 11 (4) (2003) 537–549.
[27] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, L. Cottrell, Pathchirp: Efficient available bandwidth estimation for network paths, in: Proc. Passive and active measurements workshop, 2003.
[28] M. Zangrilli, B. Lowekamp, Using passive traces of application traffic in a network monitoring system, in: High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on, 2004, pp. 77–86, doi:10.1109/HPDC.2004.1323495.
[29] A. Botta, A. Davy, B. Meskill, G. Aceto, Active Techniques for Available Bandwidth Estimation: Comparison and Application, in: Data Traffic Monitoring and Analysis, in: Lecture Notes in Computer Science, 7754, 2013, pp. 28–43.

[30] G. Aceto, A. Botta, A. Pescapè, M. D'Arienzo, Unified architecture for network measurement: the case of available bandwidth, J. Netw. Comput. Appl. 35 (5) (2012) 1402–1414, doi:10.1016/j.jnca.2011.10.010.
[31] B. Sonkoly, et al., Sdn based testbeds for evaluating and promoting multipath tcp, in: Proc. IEEE International Conference on Communications (ICC 2014), 2014, pp. 3044–3050.
[32] M. Hegde, M.K. Narana, A. Kumar, Netmon: an snmp based network performance monitoring tool for packet data networks, IETE J. Res. 46 (1–2) (2000) 15–25, doi:10.1080/03772063.2000.11416131.
[33] A. Al-Jawad, R. Trestian, P. Shah, O. Gemikonakli, Baprobsdn: a probabilistic-based qos routing mechanism for software defined networks (2015).
[34] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: Scaling flow management for high-performance networks, in: ACM SIGCOMM Computer Communication Review, 41, ACM, 2011, pp. 254–265.
[35] I. Bueno, J.I. Aznar, E. Escalona, J. Ferrer, J. Antoni Garcia-Espin, An opennaas based sdn framework for dynamic qos control, in: Future Networks and Services (SDN4FNS), 2013 IEEE SDN for, IEEE, 2013, pp. 1–7.
[36] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in sdn-openflow networks, Comput. Netw. 71 (2014) 1–30. http://dx.doi.org/10.1016/j.comnet.2014.06.002.
[37] C. Cetinkaya, Y. Ozveren, M. Sayit, An sdn-assisted system design for improving performance of svc-dash, in: Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on, IEEE, 2015, pp. 819–826.
[38] 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.
[39] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown, Reproducible network experiments using container-based emulation, in: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12), 2012, pp. 253–264.
[40] P. Megyesi, A. Botta, G. Aceto, A. Pescapè, S. Molnár, Available Bandwidth measurement in Software Defined Networks, in: Proceedings of the ACM/SIGAPP Symposium on Applied Computing, ACM, Pisa, Italy, 2016.
[41] A. Botta, A. Dainotti, A. Pescapé, A tool for the generation of realistic network workload for emerging networking scenarios, Comput. Netw. 56 (15) (2012) 3531–3547.
[42] D. Emma, A. Pescape, G. Ventre, Analysis and experimentation of an open distributed platform for synthetic traffic generation, in: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS), 2004, doi:10.1109/ftdcs.2004.1316627.
[43] A. Botta, A. Dainotti, A. Pescapé, Do you trust your software-based traffic generator? IEEE Commun. Mag. 48 (9) (2010) 158–165.
[44] D. Mills, U. Delaware, J. Martin, J. Burbank, W. Kash, Network Time Protocol Version 4: Protocol and Algorithms Specification., RFC, 5905, Internet Engineering Task Force, 2010.
[45] T. Mizrahi, Y. Moses, Time-based Updates in OpenFlow: A Proposed Extension to the OpenFlow Protocol, Technical Report, 835, Technion–Israel Institute of Technology, CCIT, 2013.
[46] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L.J. Wobker, In-band network telemetry via programmable dataplanes, ACM SIGCOMM Symposium on SDN Research (SOSR), 2015.

Péter Megyesi received his BSc and MSc in Electrical Engineering from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 2010 and 2012, respectively. Since 2012, he is a PhD student at the High Speed Networks Laboratory at the Department of Telecommunications and Media Informatics, BME. His PhD research is focused on synthetic network traffic generation. His research interests also include traffic measurements, traffic modeling and analysis and traffic identification. Since 2013, Péter is also enrolled in the Doctoral School on Innovation & Entrepreneurship organized by the EIT Digital of the European Institute of Innovation and Technology. In 2014 Péter spend six months as a visiting researcher at Traffic Group, University of Naples Federico II. Since then his main research is focused on Software Defined Networking and Network Function Virtualization. He is now also paricipating in the 5GEx H2020 EU project at BME.

Alessio Botta received the M.S. degree in telecommunications engineering and the Ph.D. degree in computer engineering and systems from the University of Naples Federico II, Naples, Italy. He currently holds a post-doctoral position with the Department of Computer Engineering and Systems, University of Naples Federico II. He has co-authored over 50 international journal (the IEEE COMMUNICATIONS MAGAZINE, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and Elsevier Computer Networks) and conference [the IEEE Global Communications (Globecom), the IEEE International Conference on Communications (ICC), and the IEEE Symposium on Computers and Communications (ISCC)] publications. His current research interests include networking, and, in particular, network performance measurement and improvement, with a focus on wireless and heterogeneous systems. Dr. Botta has served and serves as an independent reviewer of research and implementation project proposals for the Romanian government. He was a recipient of the Best Local Paper Award at the IEEE ISCC 2010. In the research area of networking, he has chaired international conferences and workshops, served and serves several technical program committees of international conferences (IEEE Globecom and IEEE ICC), and acted as a reviewer for different international conferences (the IEEE Conference on Computer Communications) and journals (the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE NETWORK MAGAZINE, and the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY).

Giuseppe Aceto is a Post Doc at the Department of Electrical Engineering and Information Technology of University of Napoli Federico II. Giuseppe received a PhD in telecommunications engineering and a MS in telecommunications engineering from the University of Napoli Federico II, Naples, Italy. His work falls in measurement and monitoring of network performance and security, with focus on censorship. Giuseppe is the recipient of a best paper award at IEEE ISCC 2010, and of ETIC AICA & Rotary International Prize for PhD Thesis on Ethics and ICT. He acted as a reviewer for different international conferences (the IEEE International Conference on Computer Communications, the IEEE International Conference on Communications, etc.) and journals (the IEEE Transactions on Computers, Future Generation Computer Systems, Journal of Network and Computer Applications, Computer Networks, etc).

Antonio Pescapé is a Full Professor at the Department of Electrical Engineering and Information Technology of the University of Napoli Federico II (Italy) where he teaches courses in Computer Networks, Computer Architectures, Programming, and Multimedia and he has also supervised and graduated more than 180 among BS, MS, and PhD students. His research interests are in the networking field with focus on Internet Monitoring, Measurements and Management and on Network Security. Antonio Pescapé has co-authored over 180 journal (IEEE ACM Transaction on Networking, Communications of the ACM, IEEE Communications Magazine, JSAC, IEEE Wireless Communications Magazine, IEEE Networks, etc.) and conference (SIGCOMM, NSDI, Infocom, Conext, IMC, PAM, Globecom, ICC, etc.) publications and he is co-author of a patent. He has served and serves as workshops and conferences Chair (including IEEE ICC (NGN symposium)) and on more than 190 technical program committees of IEEE and ACM conferences. For his research activities he has received several awards, comprising a Google Faculty Award, several best paper awards and two IRTF (Internet Research Task Force) ANRP (Applied Networking Research Prize). Antonio Pescapé has served and serves as independent reviewer/evaluator of research and implementation projects and project proposals co-funded by the EU Commission, Sweden government, several Italian local governments, Italian Ministry for University and Research (MIUR) and Italian Ministry of Economic Development (MISE). Antonio Pescapé is a Senior Member of the IEEE.

Sándor Molnár received his MSc, PhD and Habilitation in Electrical Engineering and Computer Science from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 1991, 1996 and 2013, respectively. In 1995 he joined the Department of Telecommunications and Media Informatics, BME. He is now an Associate Professor and the principal investigator of the teletraffic research program of the High Speed Networks Laboratory. Dr. Molnr has participated in several European research projects COST 242, COST 257, COST 279 and recently in COST IC0703 on “Traffic Monitoring and Analysis: theory, techniques, tools and applications for the future networks”. He was the BME project leader of the Gold Award winner 2009 CELTIC project titled “Traffic Measurements and Models in Multi-Service networks (TRAMMS)”. He is a member of the IFIP TC6 WG 6.3 on “Performance on Communication Systems”. He is a participant in the review process of several top journals and serves on the Editorial Board of the Springer Telecommunication Systems journal. He is active as a guest editor of several international journals such as the ACM Kluwer Journal on Special Topics in Mobile Networks and Applications (MONET). Dr. Molnár served on numerous technical program committees of IEEE, ITC and IFIP conferences working also as Program Chair. He was the General Chair of SIMUTOOLS 2008. He is a member of the IEEE Communications Society. Dr Molnár has more than 170 publications in international journals and conferences (see http://hsnlab.tmit.bme.hu/molnar for recent publications). His main interests include teletraffic analysis and performance evaluation of modern communication networks.