TRAFFIC MEASUREMENTS, CHARACTERIZATION AND 
EMULATION IN EMERGING NETWORKS

Péter Megyesi

Summary of the Ph.D. Dissertation

Supervised by

Dr. Sándor Molnár
High Speed Networks Laboratory
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics

Budapest, Hungary
2017
1 Introduction

In-depth understanding of the Internet traffic profile is a challenging task for researchers and a mandatory requirement for most Internet Service Providers (ISP). To this end, various methods and tools were introduced in order to help ISPs in the quest for profiling the traffic traversing through their network. With the right information in hand, ISPs may then apply different charging policies, traffic shaping, and offer different QoS guarantees to selected users or applications. Many of these methods can look at the structured information found in packet headers and create various statistical descriptors as a result, whereas others also rely on the inspection of packet payload content as well. The latter technique is called Deep Packet Inspection (DPI) and it is in the center of my Thesis. DPI matches packet payloads against a signature database which contains unique expressions for the different protocols in order to identify the application that generated the given traffic. This method is proven to be the most reliable one among the traffic classification procedures [22, 26] and often used as a ground truth for testing other classification methods [55, 16]. However, testing DPI tools in terms of both accuracy and performance is still an open issue in the research community.

During the first part of my PhD work I created a novel framework which is able to generate input for testing traffic classification tools. The User Behavior Based Traffic Emulator (UBE) [J1] can record typical user interactions with several applications on the Graphical User Interface (GUI) and construct application specific usage models which can be used later to emulate user interactions on remote controlled computers. From real network measurements UBE can also extract typical user scenarios: used applications and their share, usage patterns, etc. With these information elements, anytime when an up-to-date DPI validation trace has to be generated the user actions (e.g., mouse or keyboard events) are replayed according to the emulated user scenarios. The network traffic of the client machine is recorded and stored according to the emulated scenario. Finally, the user base is multiplied and an aggregated traffic is constructed from the recorded network traffic and the real world traffic models. The generated traffic has realistic payload and traffic characteristics both in inter-packet and user level timescales. Furthermore, the traffic created by UBE does not contain any user sensitive data and thus it can be distributed for wide audience without any privacy concerns.

The second part of my Thesis presents a method for provisioning aggregated links in access networks via a novel characterization of Internet users. In the past decade the research community gave a large attention to flow characterization. Flows have been classified in many ways, but most frequently by their size of traffic [54, 43, 27], by their duration in time [21], by their rate [35] and by their burstiness [35]. Several studies were written about the correlation between these flow behaviors [39, 41].
However, current literature lacks in profiling users in such regards. During this part of my PhD work I investigated the similar characteristics regarding Internet users. I analyzed recent measurements taken from broadband operational networks and found that while elephant users show similar packet level characteristics to elephant flows, there is a much smaller overlap between these two phenomena that one would expect. I found that only a lesser portion (10%-40%) of elephant flows are generated by elephant users and also the generation of elephant flows is not a necessary condition for being an elephant user. Taking these results further, I defined a Pareto distribution based user traffic overprovisioning model which can be used by operators to estimate the time share when the aggregated traffic of the users on the same aggregated link exceeds its capacity.

At third and final part of my PhD research I dealt with possible traffic measurement methods in Software Defined Networks (SDN). SDN is an emerging paradigm that is expected to revolutionize computer networks by offering 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 [40]. In the last few years, several proposals for measure and monitoring various networking parameters in SDN have been presented in literature. They mostly tackle problems related to bandwidth utilization [61, 33, 25, 56, 47], packet loss ratio [56], packet delay [56, 44], and route tracing [14]. All these monitoring solutions are based on approaches completely different from their 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. During this chapter I introduce a novel mechanism for measuring available bandwidth in SDN networks and validate the technique in Mininet emulation environment. Furthermore, I present analytical calculation of the measurement error due to lack of local timestamping mechanism in OpenFlow which are confirmed by the presented emulation results. Based on these results, an extension to the OpenFlow protocol providing local timestamping mechanism was proposed in order to avoid measurement errors due to network jitter.

2 Research Objectives

The first goal of my research was to create a system capable of generating traffic traces for testing purposes of traffic classification tools both in terms of accuracy and
performance. As such, the following requirements have to be fulfilled by the system:

- The traffic generation has to be automatic to the highest level of extent.

- The generated traffic has to contain up-to-date application level protocol information in the packet payloads similar what can be measured in operational networks.

- The traffic characteristics of the generated traffic e.g., bandwidth, payload sizes, packet inter-arrival times have to be similar to what are measured in operational networks.

- The users in the generated traffic e.g., parallel number of users, used applications and the way they are used has to be similar what are measured in operational networks.

- The generated traffic should be distributable among DPI testing institutes thus it must not contain user sensitive data.

- The communicating applications of the generated traffic has to be known per packet basis.

Realizing these requirements I created the architecture of the User Behavior Based Traffic Emulator. In the center of framework user behavior is emulated on remotely controlled machines and the generated traffic is captured.

Later, these individual traffic patterns are mixed together to create multi-user aggregated trace files. It was a natural progression from this point to further analyze how Internet users occupy the offered bandwidth and create an aggregated model which can tell the possibility that the total traffic exceed a given limit. The main requirement against that model is to take into account the users’ natural bandwidth scaling which is not linear. It means that if the users have 1G access bandwidth the average of the total traffic that N user generates at the same time is less than tenfold if they were to have 100M bandwidth. Moreover, the model has to take into account the three main characteristics of broadband internet traffic [38, 52]:

- The traffic proportion generated by individual users is very inhomogeneous which means that usually a small number of heavy users generate the major part of the total traffic.

- Users rarely employ the maximal bandwidth offered by their operator even in DSL networks mainly due to the usage 802.11 devices and TCP limitations.

- The average bandwidth by a single user over large time scales (e.g. in one month) is very low.
During the third part of my research my goal was to analyze novel architecture offered by Software Defined Networking and find out how it can help measuring and monitoring traffic. SDN is expected to spread widely in the upcoming years as it will be the foundation of 5G networks thus it is especially important to analyze the new possibilities and limitation in such systems. Methods for measuring network parameters such as bandwidth utilization, packet loss, and delay have been already introduced in literature, but they lacked a solution for tackling down the question of available bandwidth. Thus I attempted to fill this gap and introduced a novel mechanism for measuring available bandwidth in SDN networks. I took advantage of the SDN architecture and built an application over the Network Operating System which can measure such parameters. The method that I created can track the topology of the network and the bandwidth utilization over the links, and thus it is able to calculate the available bandwidth between any two endpoints in the network. I also validated the algorithm using the Mininet network emulation environment and multiple widely used NOS platforms such as Floodlight, ONOS and OpenDaylight.

3 Traffic Traces Frequently Used in my Research

During my PhD research an always recurring element was the application of broadband traffic traces for various analyzes. The User Behavior Based Traffic Emulator uses such measurements to define typical user behavior models, and also for evaluating the generated output by comparing it to the real traces. In the second thesis group I analyzed these real measurements for creating bandwidth utilization model for Internet users. And finally, during my third thesis group I used the these traces to replay in emulated SDN network environment in order to evaluate the created available bandwidth measurement application.

The first two measurement traces were captured in the campus network of the Budapest University of Technology and Economics (BME). The measured link was a 10Gigabit Ethernet port of a Cisco 6500 Layer-3 switch which transfers the traffic of two buildings on the campus site to the core layer of the university network. The first trace was recorded on 16:31 (CET), 18th of December 2012 and contains six minutes of traffic filtered for the wired users only. I refer to this measurement as BME Wired Trace. The second trace was recorded on 17:00 (CET), 7th of November 2013 and contains seven minutes of traffic filtered for the campus wireless users. I refer to this measurement as BME WiFi Trace. Both traces are available to the public in an anonymized format containing the following information for every packet [1]:

- Unix timestamp in seconds
- Unix timestamp in milliseconds
- Source IP address
- Destination IP address
- IP protocol number
- Source port number (in case of TCP and UDP)
- Destination port number (in case of TCP and UDP)

The third trace was measured by the Center for Applied Internet Data Analysis (CAIDA) [2] in a 10 Gbit/s backbone link between Chicago and San Jose. CAIDA periodically take measurements on this link and make them available for the research community upon request in an anonymous format (removed payload and hashed IP addresses). I analyzed multiple subsets of these data and since many results were similar I chose one given time period to present my findings. This trace was recorded on 13:15 (UTC), 20th of December 2012 and contains four minutes of network traffic. In the rest of my Thesis, I refer to this measurement as CAIDA Trace.

Finally, in Table 1 I collected the statistics for all the three traces for further reference.

<table>
<thead>
<tr>
<th></th>
<th>BME Wired Trace</th>
<th>BME WiFi Trace</th>
<th>CAIDA Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of packets</td>
<td>6804958</td>
<td>5796495</td>
<td>105444780</td>
</tr>
<tr>
<td>Number of users</td>
<td>1327</td>
<td>1970</td>
<td>680300</td>
</tr>
<tr>
<td>Number of flows</td>
<td>264117</td>
<td>307159</td>
<td>3876982</td>
</tr>
<tr>
<td>Total traffic</td>
<td>5.66 Gbyte</td>
<td>4.12 Gbyte</td>
<td>65.6 Gbyte</td>
</tr>
</tbody>
</table>
4 New Results

4.1 Research on User Behavior Based Traffic Emulation Method [J1, C1, C2]

Thesis 1. [J1]

I introduce a novel traffic generation framework which is based on user behavior emulation. This framework is unique to existing traffic generation platforms since at the one hand, is able to generate traffic simple which statistical features are close to real traffic while on the other hand, it contains real application payload data making it suitable to test Deep Packet Inspection (DPI) devices.

4.1.1 Related Work

Numerous different traffic generators were proposed in the literature in the last two decades for various testing purposes. In order to emphasis the uniqueness of the User Behavior Based Traffic Generator I collected in this subsection numerous generally known solutions which are frequently referred in the scientific literature with the topic of synthetic traffic generation.

Packet-level generators are usually used for stress testing firewalls and servers or for end-to-end performance testing. The most commonly user-space traffic generator is Iperf [6] which can generate UDP packets at a given rate or TCP packet at maximum speed. BRUTE [18] was later introduced as a kernel-level application for increasing the accuracy of the output speed rate. The same idea has been implemented for specific hardware platform (Intel IXP2400) for archiving further precision and even higher maximum output rate [17]. Other solutions, such as TG [11] or MGEN [7] supports different statistical distributions to be set up for the Inter Packet Times (IPT) and Packet Sizes (PS) information. Furthermore, Ostinato [9] is a recent generator where users can set up different streams with distinct properties and the output traffic will be the aggregate of them. Since all these solutions generates packets with dummy or random payload they cannot be used for DPI testing purposes.

Replay engines aims to reinject packets to the network as they were previously recorded with as accurate timing as possible. The most common tool for this purpose is Tcpreply [10] which is a user-space application for replying libpcap files at arbitrary speeds onto the network. The software package also includes Tcplivereplay which replays the stored traffic using new TCP connections thus adopting for the present network conditions. TCPivo [28] is a kernel-level application for traffic reply which aims to enhance the accuracy of the timing of packets critically when replaying high speed traces (e.g. recorded on OC-48 speed). Another interesting solution is
presented in [60] where authors replay OC-48 traces using multiple commodity PCs with Gigabit Ethernet network card. The collective drawback of these generators is that the measurements contain user sensitive information and cannot be distributed to other research groups for further work.

More sophisticated traffic generators are able mimic the behavior of previously recorded traces by more complex modeling of the network traffic. Harpoon [50] is a flow-based traffic generator that can mimic \textit{netflow} based measurements by analyzing various flow characteristics. Swing [58] is a closed-loop, network responsive traffic generator which is able to extract distributions for user, application, and network behavior of real measurements. Tmix [59] is a traffic emulator for ns-2 based on source-level characterization of TCP connections. Although, all these solutions can mimic the behavior of real network traffic in aspect of many different metrics, all these approaches miss to provide realistic packet payloads thus cannot be used as input for DPI devices.

D-ITG [19] is a comprehensive framework for synthetic workload generation. It’s features allows both model-based and trace-based traffic generation at the same time. The model-based mode uses Hidden Markov Model approach for modeling the IPT and PS sequence, while the trace-based mode can send packets according to the time order of a previously recorded capture file. The same two problems are present in D-ITG for DPI testing than in the previous cases: the model-based mode generates packet with synthetic payload and the trace-based mode arises privacy issues.

I created the User Behavior Based Traffic Generator such that it is unique to all these solutions as it is the only generator which can mimic the behavior of real high speed traffic measurement and can be used for DPI testing at the same time. It is also able to generate new user level measurement automatically ensuring that the database of the framework continuously contains the newest signatures of different applications providing the ground truth data as well.

4.1.2 Thesis Points

**Thesis 1.1.** [J1, C3]

\textit{I introduced the architecture of the User Behavior Based Traffic Emulator which is a traffic generation framework. The tool can mimic the behavior of an internet user, emulate it’s activity on a remote machine while recording the generated traffic with complete payload data. By multiplexing the user level traces, the framework can produce realistic traffic in arbitrary speed.}

Figure 1 presents the architecture of UBE. The designed framework is composed of the following three main components. The \textit{Measurement processor} is responsible for the definition of typical user behavior scenarios. The \textit{User emulator} can emulate
a user behavior scenario on a remote controlled machine and record the traffic generated during the process. The Traffic aggregator is able to merge multiple traffic measurements in order to create a high speed aggregated traffic mix.

In the Measurement processor the recording of the two necessary inputs are performed:

- **Recording of user interactions**: When a new application is added to the system – or one of the GUIs of the applications has changed significantly –, a user will simply use the application while its interaction with the GUI is recorded. This process typically means the naming of the input fields or buttons, not the exact location of the mouse cursor. Object names are rarely changed in a specific application thus this step is robust to version changes. The recorded typical sessions are stored in specific scripts on the test devices.

- **Traffic measurements are taken in operational broadband networks** and typical user behavior scenarios are extracted and stored in a database (for further details see Thesis 1.2). User behaviors scenarios can also be defined manually. For example, one can integrate a simple scenario of five minutes of web browsing with P2P at the background via UBE’s web interface, and the framework automatically records it to the User behavior scenario database by assigning a remote control procedure for this activity.

In the User emulator the creation of traffic segments are performed. When new up-to-date validation traffic is needed, the information from the User behavior scenario database is grabbed and user actions are emulated by remote controlling computers with the recorded user interactions. During remote controlling, GUI testing tools drive the applications on the client machine and make them to generate real traffic on the network. The generated traffic is recorded and stored in the Network.
traffic measurement database. Note that the scenarios can include such cases when the effects of the applications on each other are emulated, e.g., web browsing with streaming radio and background P2P traffic to consider the effects of the applications on each other’s traffic in the transport layer. I installed several test machines on different access network types for further increasing the diversity of the Network traffic measurement database (for further details see Thesis 1.3). The database can also store the version number of the used clients and later validation traffic for a specific snapshot in the past can be constructed.

In the Traffic aggregator the aggregation of the traffic segments is performed. The number of users is increased and an aggregated traffic is created based on the original traffic measurement and the Network traffic measurement database (for further details see Thesis 1.4). The reconstruction of per user traffic implies the arrangement of the proper measurement segments of the Network traffic measurement database according to the order defined during the identification of typical user behavior scenarios. As the operational traffic measurement and the measurements in the Network traffic measurement database have different measurement periods the packet timestamps have to be modified according to the activity period of the specific user in the user plane traffic measurement. Finally, per user traffic can be aggregated according to the timestamps.

**Thesis 1.2. [J3, C3]**

I have given a procedure for representing network traffic by character strings and an algorithm which is able to score similarities between such string patterns. The emulation framework uses this method for (i) finding typical user behaviors in real traffic measurement, and (ii) rebuilding the original aggregated traffic stream by the recorded samples in the database of the framework.

In order to crate an abstract representation of user traffic that can be used for both typical pattern analysis and similarity comparison, I proposed an algorithm that can construct string literal per user from the packet-level network traffic. In these String based traffic descriptors the used applications are represented with a character while the 1 minute granularity is signaled with delimiter characters 'x'. For example, $PxPWxPEx$ describes a three minutes long user scenario where P2P traffic was continuous, web-browsing was occurred in the second minute and e-mail in the third (see [C3] for further details on these string based traffic descriptors).

To extract typical user behavior scenarios the idea was to utilize algorithms which search for high number of occurrences with tunable soft-limit for hits and non-hits. Such algorithm was applied in [53]. In that scenario the original goal of the algorithm was to find the smallest set of signatures for the biggest coverage ratio for a specific application. My goal for the algorithm was to select the smallest set of user behavior segments for the full coverage of the total user behavior sequences.
The created similarity scoring algorithm was build around the following four principles:

- If we compare an A string to a B string, the returned score must be less than or equal to score the algorithm returns comparing the A string with itself.
- The equality must only stand if the same characters with the same amount are in both A and B.
- The algorithm must inspect the time length of the descriptor and give lesser score if it differs.
- We must have a way of setting unique values for which traffic types are suitable substitutions for each other and which are completely excluded.

Within the algorithm I defined a scoring matrix that rewards similarities and punishes differences according to these principles. I fine-tuned the values within the matrix to be specific for our use-case. For more information on this procedure see [C3].

After analyzing multiple real measurements, the algorithm created 749 different entries in UBE’s User Behavior Scenario Database. The minimum and the maximum length of these scenarios are 4 and 10 minutes, respectively. Since the user emulation is a real time process running all these scenarios takes about one week measurement on one test machine. Furthermore, the algorithm fulfilled our original expectation as the created entries can sufficiently cover the available real measurements.

**Thesis 1.3.** [C2]

*Using the User Emulator feature of the framework, I showed that running the exact same behavior scenario on different platforms (e.g. access types such as 3G, WiFi or wired and operation systems such as Android or Windows) can generate statistically distant packet level characteristics.*

The User Emulator part of UBE is able to log into remote test machines and drive its graphical user interface by emulating keystones, mouse click and other system calls. Therefore, I was able to run the exact same user scenario numerous times on different platforms in order to investigate the differences in the traffic they generate. In this thesis booklet I do not intent to present every details but rather I demonstrate the capabilities of UBE by some selected application scenarios which emphasize the capabilities of the framework. Each presented scenario was emulated at least a hundred times on the test device of UBE using every possible access type.

Figure 2a shows the traffic intensity measurement results which present the different cases that the YouTube application block send mechanism can flow. Three different test machines were used during these measurements: (i) an Android smartphone using either 3G connection (blue line), or WiFi connection at the BME campus
site (red line), (ii) a desktop Windows PC using wired connection at the BME campus site (green line), and (iii) a virtual Windows running in a server located at the National Institute of Information and Communications Technology in Tokyo, Japan (orange line). The results I measured in our campus site (green line) using a desktop Windows shows periodic and very bursty traffic. YouTube flow control shows this type of pattern if three conditions are fulfilled: (a) high speed access, (b) low round trip time and (c) no packet is lost due to congestion or buffer limit. Other Windows machines shows the normal flow of the 64 KB block sending period since a packet loss event occurred during the transmission. Android patterns show similar parameters but in case of 3G access the initial period is slower because of the limited bandwidth. These results confirm the statements presented in [15]. Figure 2b plots the Cumulative Distribution Function (CDF) of Inter-Arrival Times (IAT) of the YouTube flows. WiFi and wired measurements show the same characteristics regardless of the platform while 3G results (depicted by blue line in Fig. 2b) have 10 ms periodic tiered structure due to Node-B scheduling [31].

Figure 3a presents the CDF curve of the inter departure times of consecutive Skype packets in upstream direction. Four different test machines were used during these measurements: (i) an Android smartphone using either 3G connection (blue line), or WiFi connection at the BME campus site (red line), (ii) a desktop Windows PC using either wired connection (green line), or wireless connection (orange line) at the BME campus site, (iii) a virtual Windows running in a server located at the National Institute of Information and Communications Technology in Tokyo, Japan (brown line), and (iv) a virtual Windows running in a server located at the Federal University of Pernambuco in Recife, Brasil (purple line). It can be observed that the timing of
these packets only depend on the platform and independent on the access type. In case of native Windows the timing is very precise to the 20 ms codec frame size [36], while on virtual Windows and Android platforms differ significantly from this value. The reason for this behavior is that while timing on native Windows machines generates interrupts by accurate hardware oscillators, virtual operating systems use software interrupts generated by the host OS, which can be delayed or lost completely [24]. Authors of [24] even showed that this skewness can be used for determining the host OS of a virtual machine. Similar explanation could be behind the Android based results as in that platform applications run inside the Dalvik Virtual Machine [3] thus they don’t have direct access to the Linux kernel. Besides the voice packets, Skype also sends out sync packets after about 10 arrived frames. The timing of these frames seems random between the periodic voice packets which accounts for the linear slope at the beginning of the CDF curve.

In Figure 3b the CDF of IAT values on the other end of the conversation are plotted. This chart presents how different Internet routes can affect the downloaded stream. Measurement taken between close sites shows low jitter values, while between remote locations the skewness is higher. It is also interesting that the frame size of 20 ms overlaps the 10 ms periodicity of the Node-B scheduling using 3G access in one end of a conversation.

These measurements demonstrate how the exact same user behavior can generate statistically distant traffic patterns. This phenomenon would have been difficult to unfold without a suitable tool such as the User Behavior Based Traffic Emulator.
I showed that the User Behavior Based Traffic Emulator is able to generate multiuser traffic streams that are statistically similar to real operational broadband measurements.

The major requirement against the User Behavior Based Traffic Emulator was to be able to generate high-speed, multiuser traffic that reflects similar characteristics compared to the traffic generated by users in real measurements. In order to prove so, I carried out a performance evaluation study of the framework. The validation of traffic generators can usually be performed from different points of views and on different time-scales [C5]. I summarize my results focusing on four metrics as representative validation metrics from these important traffic characterization dimensions:

- traffic components characterization: traffic shares of applications in the aggregation,
- packet-level characterization: traffic intensity and packet size distribution,
- flow-level characterization: flow size distribution and
- scaling-level characterization: logscale diagram.

I created a constructed trace via the Traffic aggregator component of UBE using the available individual dump files in the UBE database as follows. Firstly, I consider the user level log from the BME WiFi trace used by the nDPI classificatory. This log contains the amount of data that were generated by every individual users in the aggregated measurement in a per application basis. After, I find out which individual dump file from the UBE database is the most similar to a given user using the comparison method presented in Thesis 1.2. After this step, I had a list of dump files that should be concatenated to get a similar mix to the original operational measurement. To get the final aggregated traffic I performed the reconstruction phase for every user existing in the trace. (For further details about the algorithm refer to [J1].) The main packet modifications are the following:

- Adjusting the timestamp of the packets from the measurement date to the date when the user was active. This is a fix shift and the inter-packet timers are not altered.

- Managing the IP addresses in the function of the number of emulated users. The IP addresses of the test devices in the IP header have to be altered. The framework is also capable of searching the payload of the packet for the IP address in both binary and text format and switches them for the given address. The checksums of the IP/TCP headers are also recalculated.
Figure 4: Application mix of the BME WiFi and constructed traces. The x-axis represents the traffic volumes of the top 10 applications in the BME WiFi trace whereas the y-axis represents the same traffic volumes in the constructed trace.

The constructed trace contains about 450 individual dump files, a total 3.8 GB data, 5.4 millions of packets and 217 thousands of flows. Figure 4 gives a general view about the application-space results where I plotted the traffic volume of the top 10 applications in the BME WiFi trace vs. the traffic volume in the constructed trace. Furthermore, the names of the applications are analogous to name conversation of nDPI. It can be seen in Figure 4 that the traffic shares of the top applications in the aggregation are correctly represented and important characteristics are also captured. Thus, this is a good indication that the constructed trace is suitable for accuracy testing of DPI tools since our first goal was to create a traffic mix which can generate similar amount of signature matches than the original trace would.

In order to compare traffic characteristics at packet-level the traffic intensity to downstream direction and the packet size distribution were investigated in Fig. 5a and 5b, respectively. Although, the throughput in the BME WiFi and constructed traces are not matching, the trends of the two curves in Fig. 5a show similar characteristics. This is further strengthen later in this section by a wavelet scaling analysis. In Figure 5b the CDF of packet size shows a good fit between the two curves. The shift between the two curves can be explained by a slight over-representation of small sized packets in the constructed trace. The CDF of the flow size is also well captured as depicted in Figure 5c.

To investigate the scaling characteristic of the traffic I calculated the logscale diagram for both the constructed trace and the BME WiFi trace using a popular MATLAB library for wavelet analysis [57] (for more on Wavelet analysis see [42]). The scaling characteristics for both the BME WiFi trace and the constructed traffic
Figure 5: Comparing the BME WiFi and constructed traces using different metrics

are presented by the Logscale Diagram related to the moment order \( q = 2 \) in Figure 5d. A nearly linear interval of the LD plot at octaves \( 4 \leq j \leq 11 \) can be observed for both traces revealing the well-known Long-Range Dependence (LRD) property of the aggregated traffic [42]. A linear regression to this interval gives an estimation of LRD parameter of \( H_{BME\text{WiFi\,trace}} = 0.875 \) and \( H_{\text{constructed\,trace}} = 0.842 \) for the original measured and the emulated traffic, respectively. These results clearly indicate that the emulated traffic accurately captures the complex scaling structure of the original measured traffic.

In summary my validation study showed that the User Behavior Based Traffic Emulator is able to reproduce an aggregated traffic which captures the characteristics of the original measurements.
4.1.3 Applicability of Results

The main purpose of the User Behavior Based Traffic Emulator was to be able to create high-speed realistic traffic traces that contains up-to-date application signatures therefore it is suitable to test Deep Packet Inspection tools. The validation procedure that I presented proves that UBE fulfills that purpose. A graduating student in our group used these traffic traces to evaluate five different traffic classification algorithms: a port-based classifier (based on IANA port numbers [5]), SPID [4], TSTAT [12], OpenDPI [8] and Captool (proprietary classifier of Ericsson). The work identified that Captool is the most reliable classification tool from the investigated ones, but OpenDPI provides good results also. The others performed significantly worse, the port-based classification being the most unreliable from them.

In 2016 we also shared UBE’s traffic database with Valentín Carela-Español and Pere Barlet-Ros from UPC Barcelona Tech. Their group is one of the most active in the field of evaluating traffic classification tools having numerous fundamental publications, e.g. [55] and [23]. By their work the User Behavior Based Traffic Emulator can fulfill its original purpose as being used by independent research institute for evaluation the performance of classification tools.

4.2 Pareto Characterization of Internet Users

Thesis 2. [C6, C7]

I introduce a novel traffic characterization method in which user bandwidth utilization can be overprovisioned by a single Pareto distribution. Based on the parameters of the Pareto distribution I give a formula that operators can use to estimate the time share when the aggregated traffic of the users on a given link exceeds its capacity.

4.2.1 Related Work

In the past decade the research community gave a large attention to flow characterization, whereas similar works are not present for user characterization. Flows has been classified by their size of traffic (as elephant and mice) [54, 43, 27], by their duration in time (as tortoise and dragonfly) [21], by their rate (as cheetah and snail) [35] and by their burstiness (as porcupine and stingray) [35]. Several studies were written about the correlation between these flow behaviors [39, 41].

There are several different definitions for elephant flows in the literature. In [43] authors propose two techniques to identify elephants. The first approach is based on the heavy-tail nature of the flow bandwidth distribution, and one can consider a flow as an elephant if it is located in this tail. The second approach is more simple, elephants are the smallest set of flows whose total traffic exceeds a given threshold.
Estan and Varghese [27] used a different definition. They considered a flow as an *elephant* if its rate exceeds the 1% of the link utilization.

However, the definition given by Lan and Heidemann [35] become a rule of thumb in later literature (e.g. both [22] and [41] use this definition). They define *elephant flows* as flows with a size larger than the average plus three times the standard deviation of all flows. They use the same idea for categorize flows by their duration, rate and burstiness as *tortoise*, *cheetah* and *porcupine*, respectively. [35] was also the first study that presented the *cheetah and snail* and the *porcupine and stingray* classifications. *Tortoise and dragonfly* properties of traffic flows were first investigated in [21]. Here, the authors considered a flow as *tortoise* simply if its duration was larger than 15 minutes. Given the generality and the rule of thumb nature of the definition by Lan and Heidemann [35] I will use the same definition for *elephants* later in my work.

In [49] Sarvotham et al. present a comprehensive study that traffic bursts are usually caused by only few number of high bandwidth connections. They separate the aggregated traffic into two components, *alpha* and *beta* by their rate in every 500 ms time window. If the rate of the flow is greater than a given threshold (mean plus three standard deviations) than the traffic is *alpha*, otherwise it is *beta*. Authors determine that while the *alpha* component is responsible for the traffic bursts, the *beta* component has similar second order characteristics to the original aggregate.

However, current literature lacks in profiling users in such regards. The term *elephant user* appears in [37] where the authors calculate the Gini coefficient for the user distribution. The Gini coefficient is usually used in economics for measuring statistical dispersion of a distribution. They calculate the value of the Gini coefficient for the distribution of the number of bytes generated by the users as 0.7895 but no further discussion is presented. In [45] authors investigate application penetration in residential broadband traffic. They calculate the results separately for the top 10 *heavy-hitters* (the top 10 users that generated the most traffic) in their measurement data. Besides pointing out the fact that the majority of the data is generated by a small group of users the paper does not tackle any further issues about *elephant users*.

In my work I focused on user characterization by defining *elephant users* in a similar manner than *elephant flows*. However, I showed that the two phenomena have much less overlap than one would anticipate. Furthermore, I build up a user model based on the analyzed traffic and give an general formula for bandwidth utilization by aggregated users.
4.2.2 Thesis Points

**Thesis 2.1. [C6]**

I have analyzed the elephant-mice phenomenon in real operational measurements and pointed out that while elephant user and elephant flow shows similar aggregated statistics, the overlap between the two phenomena is only 10-40%.

In order to unwrap the phenomenon of elephant users I analyzed multiple real measurement traces from which I will present the result on the BME Wired and CAIDA traces. In Figure 6a and 6b the traffic of elephant users and elephant flows are plotted against the original traffic in the BME Wired and CAIDA traces, respectively. The relative differences to the original traffic are also presented. In case of the BME
Figure 7: The number of elephant and non-elephant flows generated by elephant users

Wired trace both the elephants users and elephants flows are responsible for sufficient amount of the total traffic (80%-85%), while in the CAIDA trace this ration is a bit smaller (60%-70%). Since the traffic of elephants follows the bursts in the original traffic (the relative differences are also smaller at these peaks), the results suggests that elephant users are main cause for traffic burstiness. The relative difference between the throughput curves of elephant users and elephant flows is very small in case of the BME Wired trace (2%-4%), whereas it is a bit higher in the CAIDA trace (8%-12%).

Inter arrival time between consecutive packets are presented in 6c and 6d in the BME Wired and CAIDA traces, respectively. The curves show similar characteristics for elephant users and elephant flows. The CDF curves of elephants are increasing slower than the original aggregate’s which is an expected behavior since traffic of elephants are the rarefaction of the original packet stream. Also, the figures show very slight difference between the CDF curves of elephant users and elephant flows.

On the other hand, in Figure 7 every dot represents an elephant user according to the generated number of elephant flows and mice flows. These results indicate that there is no correlation between the number of elephant flows and mice flows generated by an elephant users. Furthermore, a user can be an elephant without generating any elephant flows. There was 53 elephant users in the CAIDA trace who did not generated any elephant flow. They account for 8% of all elephant users in the CAIDA trace. In the BME Trace this number is only 3, but since there were only 56 elephant users in that measurement their share is 5%.

Another interesting result is that in case of the CAIDA Trace only the 9.13% of elephant flows were generated by elephant users. In case of the BME Trace this value is higher, namely 37.85%. These result clearly indicate that the overlap between the elephant user and elephant flow phenomenons could be much smaller in some cases that one would expect.
Table 2: Values after Pareto fitting in the BME Wired and the CAIDA traces

<table>
<thead>
<tr>
<th>Window size</th>
<th>BME Wired trace</th>
<th>CAIDA trace</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$X_m$</td>
</tr>
<tr>
<td>100 ms</td>
<td>0.90</td>
<td>250 kbps</td>
</tr>
<tr>
<td>10 ms</td>
<td>0.89</td>
<td>831 kbps</td>
</tr>
</tbody>
</table>

Thesis 2.2. [C7]
I showed that in independent broadband operational measurements the user bandwidth utilization can be modeled by a Pareto distribution with proper parameter settings. Based on these results, I proposed a dimensioning method which can be used for over-estimating the bandwidth utilization of Internet users.

In the proposed user model, I approximated users’ bandwidth utilization by a Pareto distribution. To confirm the viability of this assumption I plotted the Complementary Cumulative Distribution Function (CCDF) of bandwidth utilization values by individual users in 10 ms and 100 ms time windows in Figure 8 in the BME Wired and CAIDA traces. Though, the nature of these two measurements fundamentally differ from each other, the corresponding results give confidence for using Pareto distribution for modeling user bandwidth utilization.

In the created user model I used a modified version of the Pareto distribution, where $S_m$ represents a maximum value that the distribution can take. This value represents a maximum allowed bandwidth that a user cannot exceed. In this case the distribution function can be expressed as follows:

$$
\hat{F}_x(x, S_m, X_m, \alpha) = \begin{cases} 
0 & \text{if } x < X_m \\
1 - (\frac{X_m}{x})^\alpha & \text{if } X_m \leq x \leq S_m \\
1 & \text{if } x > S_m 
\end{cases}
$$

(1)

After fitting the Pareto distribution to the given measurements (shown in Figure 8) I collected the resulted $\alpha$ and $X_m$ values in Table 2. The values also show close result for the BME Wired and the CAIDA traces even though they were measured in very different network types. These measurements strengthens the generality of the given model. Furthermore, the results also strengthens that the larger the base of the time averaging is, the lower the $X_m$ value will be.

Thesis 2.3. [C7]
I gave a formula that operators can use to estimate the capacity of an aggregated link.
Figure 8: Pareto distribution fitting for user bandwidth utilization in BME Wired and CAIDA traces for a given $\epsilon$ availability level based on the Pareto overprovisioning. Moreover, the parameters in the formula can be easily adjustable by new broadband measurements as the Internet traffic keeps growing in an exponential rate.

The modification in Eq. 1 compared to the original Pareto distribution also allows us to calculate with the measured shape parameters $\alpha < 1$ since the expected value in this case is not infinite. Thus the expected value of the modified Pareto distribution can be calculated as follows:

$$M = \int_{X_m}^{S_m} f_x x \, dx + (1 - \hat{F}_x(S_m)) S_m = \frac{\alpha}{\alpha - 1} X_m - \frac{\alpha}{\alpha - 1} \frac{X_m^\alpha}{S_m^{\alpha - 1}} + \left( \frac{X_m}{S_m} \right)^\alpha S_m$$

(2)

Also, using the well-known fact that the variance is equal to the expected value of the square of the distribution minus the square of the mean of the distribution, the following formula gives the variance:

$$\sigma^2 = \int_{X_m}^{S_m} f_x x^2 \, dx + (1 - \hat{F}_x(S_m)) S_m^2 - M^2 = \frac{\alpha}{\alpha - 2} X_m^2 - \frac{\alpha}{\alpha - 2} \frac{X_m^\alpha}{S_m^{\alpha - 2}} + \left( \frac{X_m}{S_m} \right)^\alpha S_m^2 - M^2$$

(3)

For giving an availability model from these I applied the Central Limit Theorem for the sum of independent and identically distributed random variables. That way one can calculate the possibility $\varepsilon$ that aggregated traffic of $N$ users exceeds a given
Usually, a service provider is interested in the capacity that should be used for the aggregated link ($C_R$), for a given user number $N$ and an availability rate $\varepsilon$. Thus I inverted the formula as follows:

$$C_R = NM + \sqrt{2N}\sigma \text{Erfc}_{inv}(2\varepsilon)$$  (4)

### 4.2.3 Applicability of Results

The goal behind creating such user model was to give a general formula that internet service providers can use during provisioning their access networks. Today passive optical (PON) solutions provide the largest residential access bandwidth for users, thus ISPs prefer to deploy such networks over xDSL and DOCSIS technologies. To this end, having appropriate knowledge about which type of PON technology is the most sufficient for a given access speed is crucial for the service providers.

In our work, we were able to apply my model for such analysis [C7]. Using the formula given in Eq. 4 and general industrial cost models of Time Division Multiplexing (TDM) PON and Wavelength Division Multiplexing (WDM) PON, they were able to identify an inflexion point between the two technologies. Their analysis showed that if an ISP want to offer less than 600 Mbps of access bandwidth for every user in a PON network TDM-PON has lower per user capital cost, whereas above 600 Mbps they should deploy WDM based PON networks.

### 4.3 Traffic Measurements in Software Defined Networks

**Thesis 3.** [J2, C8]

I introduce a novel method for end-to-end available bandwidth measurement in Software Defined Networks based on the features provided by the Network Operating System. I also show that the error of such measurements are unavoidable with the current OpenFlow protocol specification and gave analytical analysis on how to calculate it.

#### 4.3.1 Related Work

Software Defined Networking (SDN) is an emerging paradigm that is expected to revolutionize computer networks. With the decoupling of data and control planes and the introduction of open communication interfaces between layers as presented in Fig. 9, SDN enables programmability over the entire network, promising rapid innovation in this area. In the recent years, there has been several proposals for monitoring Software Defined Networks since the architecture provides novel abstractions to access information about the network.
FlowSense authors [61] 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 [25]. Other papers propose to use the FlowStatsReq message in OpenFlow to poll the interface and flow counters in the switches for bandwidth measurement [25, 56, 47]. Furthermore, PayLess [25] and MonSamp [47] 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 [56] offers a solution for loss and delay monitoring as well. For measuring packet loss, it polls the flow counters on the ingress and egress switches for a given flow and calculates the difference. For measuring network delay, it uses the SDN controller to inject probe packets into the network along a given path and then loop them back to the controller. The tool is able to calculate the delay for the given path using the round trip time between ingress and egress switches. Phemius and Bouet [44] 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. However, the current literature lacks a solution for measuring available bandwidth in SDN.

Available bandwidth is an important dynamic characteristic of a network path,
being equivalent to the amount of traffic that can be added to the path without affecting the other flows that traverse part of it, and independently from their bandwidth-sharing properties. Such definition tells it apart from other bandwidth-related metrics such as bulk transfer capacity and from the maximum achievable throughput [46]. In traditional networks, available bandwidth estimation techniques are typically classified into active and passive. Passive techniques use multiple measurement points in the network to monitor bandwidth utilization, packet loss ratio, and packet delay. These techniques are very complex to deploy in traditional networks thus they are rarely used in practice. Active techniques send probe packets into the network and analyze how network traversal affected their spacing/arrival to infer network status. On the basis of the hypotheses on the analyzed path and on the type of probing procedure adopted, active ABW estimation techniques in the literature can be referred to two models: probe gap and packet rate. Probe gap tools such as Spruce [51] or Traceband [29] 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 [32] and PathChirp [48].The performance of most of active ABW estimation tools currently available is scenario-dependent and require non-trivial calibration [20, 13]. In my work I used a passive technique taking advantage of the NOS in the architecture of SDN. I used the northbound API to query the topology of the network and to monitor the bandwidth utilization of the links. With this information the given algorithm can calculate the available bandwidth for any path in the network in any given time.

However, the new possibilities provided by SDN and its NOS layer introduce new issues, limitations, and sources of error, which were previously undiscussed in such manner. This also effects the proposed ABW measurement algorithm as well as the other related works presented in this section. Therefore, I also analyzed the source of this error and gave insights in how to model it.

**Thesis 3.1.** [C8]

*I gave a novel method for end-to-end available bandwidth measurement in Software Defined Networks based on the features provided by the Network Operating System.*

Using the northbound API of the NOS the algorithm query the topology abstraction of the network which is a mandatory feature in every SDN controller [34]. Firstly, the 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 Tab. 3). 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 I used an approach similar to the one previously presented in [25, 56, 47]: periodically
poll the counters in the SDN switches using the FlowStatsReq 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, the algorithm calculates the available capacity \( a_i \) on every link in the network. Based on the \( a_i \) values one can calculate the available bandwidth on a given path \( P \) through the following equation

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

The defined 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 the algorithm has to find out its route on the network, and then calculate the available bandwidth using Eq. (5). I used 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:

\[
ABW_{A \rightarrow B} = \max_{P \in P_{A \rightarrow B}} \min_{e_i \in P} a_i. \tag{6}
\]

For solving this equation I defined a modified Dijkstra algorithm where the metric of a path is not measured by the sum of the edge capacities (distances) but by Eq. (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).

**Table 3: Notation list.**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G(V, E) )</td>
<td>the directed graph representation of the network topology with node set ( V ) and edge set ( E )</td>
</tr>
<tr>
<td>( e_i )</td>
<td>( i^{th} ) link in the network topology graph</td>
</tr>
<tr>
<td>( c_i )</td>
<td>the capacity of ( e_i )</td>
</tr>
<tr>
<td>( b_i )</td>
<td>the current bandwidth load on ( e_i )</td>
</tr>
<tr>
<td>( a_i )</td>
<td>the available capacity on ( e_i ), ( a_i = c_i - b_i )</td>
</tr>
<tr>
<td>( P_{A \rightarrow B} )</td>
<td>the set of all available paths from ( A ) to ( B )</td>
</tr>
</tbody>
</table>
3. **Multipath scenario.** In this case multiple paths can be used between two points in the network. Thus, the task turns into 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).

For validating the available bandwidth algorithm I conducted extensive network emulation scenarios. As it can be observed in Figure 10 the created algorithm captures well the reference values. During the measurements I used three different traffic conditions as follows:

- **Constant Bit Rate (CBR) Traffic** (Figure 10a) - In this case two sources generates CBR traffic to two different destinations. The first one generates 4 Mbps of UDP traffic for 100 seconds, than the source sleeps for 100 sec (generating no traffic) and restarts sending with 8 Mbps rate. The second source sends 10 Mbps of UDP traffic 100 sec after the start of the measurement until the end. In this case the created SDN application captures well the reference values.

- **Variable Bit Rate (VBR) Traffic** (Figure 10b) - For a more dynamic scenario, in this case the two sources generated traffic using Pareto distribution for the inter-departure times of packets. I used $\lambda = 1.75$ as shape parameter for both source, whereas for scale parameter I set $X_1 = 1ms$ and $X_3 = 0.5ms$ for the first and second source, respectively. In this case the error increases, but still close to the reference values.

- **Real Traffic** (Figure 10c) - For realistic traffic generation, in this case I used the BME WiFi trace to replay. The algorithm still captures well the reference values with similar error rate than in case of VBR traffic.

**Thesis 3.2. [J2]**

I showed the main characteristics which determine the accuracy of counter based bandwidth measurements in Software Defined Networks. These are: (i) measurement overhead, (ii) accuracy limitation for lack of synchronization, (iii) critical time-scale dependence of estimation, and (iv) accuracy limitation for lack of timestamp.

**Measurement overhead**

In traditional networks the measurement overhead caused by passive methods has been subject of several studies and proposals [25, 47]. Due to both the architecture of SDN networks and the different possibilities for monitoring it provides, measurement overhead can have multiple 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 I proposed, 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.

**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. 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 $\delta_{\text{max}}$ 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

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

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

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

where $\tau$ is the polling rate, and $L_1$ and $L_2$ are the counter values in the first and second measurements, respectively. The maximum delay in polling $\delta_{\text{max}}$ in turn depends on the number of switches to be polled $N$, the capacity of the link connecting the controller to the control network $C_{\text{ctl}}$ and the length of the query packet $\text{Len}_{\text{query}}$:

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

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 $C = C_{\text{ctl}}$, one can obtain

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

Given the size of FlowStatsReq message $\text{Len}_{\text{query}} = 56\text{Bytes}$, a controller managing $N = 101$ switches, with poll rate $\tau = 0.5s$ can suffer up to 44.7Kbps 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, more frequent polling rates results in higher measurement error.

**Critical time-scale dependence of estimation**

For its very definition, ABW is a metric that depends on a time interval, thus its dependence on the choice of the *averaging timescale* $\tau$ 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 seconds [30], while techniques provides a

---

$^1$OpenFlow Switch Specification Version 1.5.1 Section 7.3.5.2
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 \( \delta_{\text{ingr-egr}} \) the maximum delay for a packet to traverse the SDN network, and \( \tau \) the polling period, this condition translates in \( \tau \gg \delta_{\text{ingr-egr}} \).

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.

**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. If we mark the time as \( T_{GT}^i \) when an \( L_i \) counter value is read in the \( i \)th measurement period in a given OpenFlow switch, the monitoring packet takes some time traversing through the network to reach back to the SDN controller. Here, the ABW application also timestamps the packet with \( T_{meas}^i \) which inherently differs from the original timestamp value. By defining as \( \Delta L \) the difference between the counter values \( L_i \) and \( L_{i+1} \) in the \( i \)th and \( i+1 \)th measurement periods, respectively, the following formula can be obtained for the error:

\[
\epsilon = \frac{\Delta L}{T_{GT}^{i+1} - T_{GT}^i} \frac{T_{meas}^{i+1} - T_{meas}^i}{\Delta L} = \frac{1}{T_{GT}^{i+1} - T_{GT}^i} \frac{T_{meas}^{i+1} - T_{meas}^i}{1}
\]

(7)

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

\[
\epsilon = \frac{\tau}{\tau + \delta_{i+1} - \delta_i} - 1 = -\frac{\delta_{i+1} - \delta_i}{\tau + \delta_{i+1} - \delta_i}
\]

(8)
Table 4: Error rate of ABW measurements using real traffic replayed by D-ITG with no added delay.

<table>
<thead>
<tr>
<th>Polling period in seconds</th>
<th>[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>Mean</td>
<td>0.72</td>
</tr>
<tr>
<td>STD</td>
<td>0.94</td>
</tr>
</tbody>
</table>

**Thesis 3.3. [J2]**

I showed that in Software Defined Networks measurements using the OpenFlow protocol inherently introduces an error to the system due to the design of the protocol. I also showed that this error does not depend on the extent of the delay between the Network Operating System and the SDN switches, but rather the distribution of the difference between the delay of two subsequential measurement packets (hence the network jitter on the control channel).

In order to fully understand the measurement error caused by the lack of local timestamping at the SDN switches, I conducted several Mininet emulation scenarios using a wide range of network delay and polling rate setups. Firstly, I set up the network on an as it is basis which can be considered as an ideal case, i.e. the one in which timestamping is done in the switches at the moment the response packet is put on wire. Table 4 reports the mean and the standard deviation of the measured error values for such cases. However, it can be noticed 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 the values averaged on a larger time scale while the difference in the timestamp approximation remains the same, thus having a smaller relative effect. One could average the values from a more frequent measurement in order to get a more precise on a larger timescale but the error is similar to using a larger polling period which generates less overhead. If one considers the minute time scale, where typically the current ABW tools operate [13], the average error rate is less than $10^{-5}$ which can be considered as very accurate.

As a next step, I introduced artificial delay between the SDN switches and the SDN controller to investigate its effect over the error rate. In details, I 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 in various combinations. I used fifteen different mean-std delay setups for the five lowest polling periods in Table 4 and emulated every scenario four different times. In Fig. 11 I plotted the...
result of every measurement using real traffic replay. The measured mean delays in all the 300 measurement cases are depicted against the mean and the standard deviation of the delay in Fig. 11a and 11b, respectively. The results in Fig. 11a confirms 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. 11b is a clear indication that in case of normal delay distribution there is a linear dependence between the standard deviation of the monitoring packet delay and the mean error of the available bandwidth measurement.

Furthermore, if we assume that the network delay has normal distribution $\mathcal{N}(m, \sigma^2)$, based on Eq. 8 the distribution of the error can be calculated as the difference of two independently and identically distributed normal variable. Thus in this case the distribution of the difference of two consecutive values will be $\mathcal{N}(0, 2\sigma^2/\tau)$, thus the distribution of the error will be $\mathcal{N}(0, 2\sigma^2/\tau)$, if $\tau \gg \sigma$. Based on this formula one can calculate the mean of the absolute error as follows:

$$E(|\epsilon|) = \int_{-\infty}^{\infty} \left| \frac{y}{\tau - y} \right| f_y(y) dy = \int_{-\infty}^{\infty} \frac{y}{\tau - y} f_y(y) dy + \int_{0}^{\infty} \frac{y}{\tau + y} f_y(y) dy$$

$$= \int_{0}^{\infty} \frac{2\tau y}{\tau^2 - y^2} f_y(y) dy \quad \tau \gg \sigma$$

$$= \sqrt{\frac{4 \sigma}{\pi \tau}}$$

Figure 11: The average error measured during real traffic replay using different mean delay and standard deviation setups.
Eq. 9 also confirms the measurement result presented in Fig. 11b where the error show a liner connection with the ratio between $\sigma$ and $\tau$ with a factor of $\sqrt{\frac{2}{\pi}}$.

4.3.2 Applicability of Result

Based on the work I’ve done in the field of SDN monitoring, together with my colleagues we proposed to extend the OpenFlow protocol with additional timestamping mechanism in order to lower the rate of monitoring measurement error that would otherwise unavoidable. In particular, we proposed two different 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 proposed 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.

5 Summary

During my PhD work I dealt with various aspects of measuring, characterizing and emulating network traffic for emerging scenarios. I started by creating a novel traffic generation framework called the User Behavior Based Traffic Emulator. I showed that by emulating user activity on remotely controlled machined one can build up a traffic database that contains up-to-date application signatures without any privacy concerns. By the right algorithm of aggregating these elements UBE is able to create an input traffic stream for testing Deep Packet Inspection tools in terms of both performance and accuracy. Therefore the presented framework is quite unique on the palette of numerous traffic generation tools.

Continuing with the characterization of user behavior I created a model where user bandwidth utilization can be provisioned by a modified Pareto distribution. Using this model I created a formula which can be used by operators to estimate the time share when the aggregated traffic of their users exceeds the capacity of the aggregated link. This model has the advantage that it takes into account the users’ natural bandwidth scaling as well as the maximum bandwidth they’re offered. Moreover, the model’s parameters can be easily adjustable by new broadband measurements as the generated traffic keeps growing in an exponential rate.
During the last part of my research I analyzed architecture of Software Defined Networks and I gave an algorithm for measuring available bandwidth in such networks. I showed that the novel architecture that SDN offers also introduces new issues, limitations, and sources of error, which were previously undiscussed in such manner. I provided an analytical modeling of the issues and also, an experimental evaluation in Mininet emulation environment. Based on my work, we proposed to extend the OpenFlow protocol with additional timestamping mechanism in order to lower the rate of monitoring measurement error that would be otherwise unavoidable.

Acknowledgements

I wish to thank Dr. Sándor Molnár, my PhD supervisor for his support during my student years. Dr. Géza Szabó, research fellow at Ericsson Hungary was supervising the work that I presented in Section 4.1. His help was a fundamental element in achieving the presented results. Furthermore, the work presented in Section 4.3 was done in cooperation with the Traffic Research Group, University of Naples Federico II. Namely, I wish to thank Alessio Botta, Giuseppe Aceto and Antonio Pescapé for their support.
References


Publications

Journal papers (11.3 p)


Conference papers (13.5 p)


[C7] J. Czekus, P. Megyesi, A. Mitcsenkov, D. Mazroa Hardware cost and capacity analysis of future TDM-and WDM-PON access networks In *Proc. of the 16th International Conference on Transparent Optical Networks (ICTON)*, Graz, Austria, 6-10 July 2014. (3/4=0.75 p)


Independent Citations


