Budapest University of Technology and Economics
Department of Telecommunications and Media Informatics

Traffic Measurements, Characterization and Emulation in Emerging Networks

Péter Megyesi

PhD Dissertation

Advisor:

Dr. Sándor Molnár (Associate Professor)

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

Budapest, Hungary
2017.

...For Those About to PhD, I Salute You

Abstract

Understanding the traffic profile of the Internet has become one of the most important and most challenging issues for Internet Service Providers (ISP). To this end, ISPs use different traffic identification tools that can help them profiling the network applications which load their systems. Using this crucial information they are able to apply different charging policies, traffic shaping, and offer different Quality of Service (QoS) guarantees to selected users or applications.

Traffic identification tools are usually classified into three groups: (i) port-based methods, where a protocol is identified only by the information in the transport-layer header, (ii) statistical-based methods where statistical indicators are derived from different characteristics of the network flows (e.g., average of packet sizes or inter-packet times) and the results are matched with a learned database, and (iii) Deep Packet Inspection (DPI) where packet payloads are matched against a signature database which contains unique expressions for the different protocols. DPI was introduced as a solution for several issues associated with port-based and statistical-based identification approaches in order to achieve an accurate and timely traffic identification. This procedure was proven to be the most efficient one among the traffic identification methods and it is often used as a ground truth for testing other identification tools. However, testing DPI algorithms in terms of both performance and accuracy is still an open issue in the research community.

The goal of this dissertation is to present my research achievements on a traffic emulation framework named the User Behavior Based Traffic Emulator (UBE). UBE is able to generate high speed traffic aggregates mixed up by arbitrary user profiles. By this process the framework can generate a suitable input for DPI testing which can also be freely distributed among the research community without any privacy concerns. In the first part I present the architecture of UBE which is constructed

by three main parts: (i) the user emulator where an arbitrary user behavior can be played in remotely controlled machines, (ii) the measurement processor where typical user behaviors can be extracted from real traffic measurement, and (iii) the traffic aggregator where high speed aggregates can be mixed up using the recorded traffic traces. I highlight the new results and explain how the three parts can work together in order to achieve my original research goal. I validate the framework by comparing its output with traffic measurements on real operational networks.

In the second part of this Thesis I continue my research with the characterization of user behavior by creating a model where user bandwidth utilization can be provisioned by a modified Pareto distribution. Using this model I gave 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.

In the final part of my dissertation I follow the newest trends in networking by investigating the advantages of centralized control and OpenFlow protocol in Software Defined Networking (SDN) for traffic measurement purposes. I defined an algorithm that can determine the available bandwidth between any two points in a SDN network. I also analyzed the error rate of such measurements and gave theoretical error bounds based on the network delay and the measurement time interval.

Kivonat

Az Internet forgalmának mély megértése különösen fontos feladat minden Internet szolgáltató számára, mely egyben nagy kivások elé is állítja őket. Annak érdekében, hogy a szolgáltatók a lehető legpontosabb képet kapják a hálózatukon átmenő forgalomról, különböző forgalom osztályozó eszközöket szokás alkalmazni, melyek képesek megmondani egy adott hálózati folyamról, hogy milyen alkalmazás generálta őket. Az ilyen információk segítségével a szolgáltatók képesek lehetnek egy adott forgalom típust vagy felhasználót más szolgáltatásminőségben kezelni és így más számlázási irányelvet is alkalmazni.

A forgalom osztályozó eszközöket három típusba szoktuk besorolni: (i) port alapú módszerek, ahol egy adott alkalmazást a szállítás-rétegbeli portszámok alapján azonosítanak, (ii) statisztikus módszerek, ahol egy adott folyam különböző tulajdonságaiból leírókat képeznek (pl.: átlagos csomagméret vagy csomagok közötti időintervallumok), és ezeket hasonlítják össze egy tanított halmazzal, illetve (iii) úgynevezett Deep Packet Inspesction (DPI) eszközök, ahol a csomag tartalmában keresnek az adott alkalmazásra jellemző egyedi regurális kifejezéseket. A DPI alapú eszközök azért lettek nagyon népszerűek az elmúlt évtized során, mert pontos és időszerű megoldást tudtak adni olyan alkalmazások forgalmának a felismerésére, ahol a port és statisztikus alapú eszközök nem tudtak jól teljesíteni. A csomagtartalomra való mintaillesztés bizonyítottan nagyobb pontossággal tudta azonosítani a hálózati alkalmazásokat, ezért sok esetben ilyen módszert használtak referencia bemenet előállítására más forgalomazonosító eszközök tesztelése során. Ennek ellenére a DPI eszközök pontos tesztelése mind pontosság, mind teljesítmény szempontjából a mai napi megoldatlan probléma a hálózati kutatók között.

A doktori kutatásom során az első lépés az úgynevezett Felhasználó Viselkedés

Alapú Forgalom Emulátor (User Behavior Based Traffic Emulator – UBE) megtervezése és megépítése volt. A keretrendszer képes nagysebességű, aggregált forgalmi mintákat előállítani tetszőleges felhasználói profil alapján. A rendszer által generált kimenet alkalmas DPI eszközök tesztelésére (mind pontosság, mind teljesítmény tekintetében), illetve szabadon terjeszthető a kutatói közösség között, hiszen nem tartalmaz semmilyen privát adatot. A disszertációm első harmadában részletesen bemutatom a keretrendszer architektúráját, mely az alábbi három elemre épül: (i) *felhasználó emulátor*, amely képes tetszőleges felhasználói viselkedést eljátszani távolról irányított tesztgépeken, (ii) *mérés feldolgozó*, amely valós forgalmi mérések elemzésével határoz meg tipikus felhasználó viselkedéseket, illetve (iii) *forgalom aggregátor*, amely az egyedi forgalmi minták összefűzésével képes nagysebességű és valósághű forgalmi minták előállítására. A dolgozatban kiemelem az elért főbb eredményeket megmutatva, hogy a keretrendszer három eleme hogyan működik együtt annak érdekében, hogy elérjem az eredeti célkitűzésemet. Ezen felül a keretrendszer által előállított kimenetet összehasonlítom valós forgalmi adatokkal, hogy ezen keresztül is igazoljam a rendszer helyes működését.

A doktori dolgozatom második harmadában mélyebben elemzem a felhasználók által generált forgalmat. Ennek során először megmutattam, hogy az Internet felhasználók ugyan hasonló módon karakterizálhatok, mint a hálózati folyamatok, de e két jelenség között még sincs akkor átfedés, mint ahogy arra előzetesen számítani lehetett. Ezután az eredményekre egy olyan modellt illesztettem a felhasználók által kihasznált sávszélességre, ahol egy módosított Pareto eloszlással ez az érték könnyen felülbecsülhető. Továbbá, a modellből levezettem egy olyan képletet, melyet Internet szolgáltatók könnyen tudnak használni, amikor hozzáférési hálózatok tervezésénél több felhasználó által megosztott közös link szükséges méretét becsülik.

Végezetül, a disszertációm utolsó harmadában — követve a hálózati világ legújabb trendjeit – az OpenFlow protokoll és a Szoftverizált Hálózatok (SDN) által nyújtott új lehetőségeket vizsgáltam meg, forgalmi mérések szempontjából. Definiáltam egy új algoritmust, mely képes egy SDN hálózatban bármely két pont között meghatározni az aktuális rendelkezésre álló sávszélességet. Ezen felül megmutattam, hogy az ilyen jellegű mérések mindenképpen egy bizonyos hibafaktorral működnek, melynek oka az SDN architektúra felépítéséből származik. Ezt a hibát analitikusan is megvizsgáltam és az eredményeket igazoltam kiterjedt mérési eljárások segítségével.

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 Chapter 2 His help was a fundamental element in achieving the presented results. Furthermore, the work presented in Chapter 4 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.

Contents

Chapter 1

Introduction

Network connectivity has become such a fundamental element in our society that the diversity and complexity of the Internet far exceeded the expectations of the original designers. This has led to a situation where in-depth understanding of the Internet traffic profile is a very 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. One of the most crucial profiling type is *traffic classification*: to associate a network flow to the application (or application type) that generated it. In the past decade the research community has offered numerous methods and techniques to give a solution for this problem [57, 92]. Parallel to that many commercial traffic classification platforms have been introduced lately in order to fill this market niche. Nevertheless, such commercial platforms are just one element in the dynamic increase of the costs for improving the performance of carrier networks. As a result, ISPs are somewhat forced to use the information yield by such classification tools to apply different charging policies, traffic shaping, and offer different QoS guarantees to selected users or applications in order to increase their return on investment. All these trends have given a major boost to scientific publications in the field of traffic classification, and many research groups started to focus on the topic as well [65].

1.1 State-of-the-art in Traffic Classification

During the early days of the Internet new applications registered a unique transport layer (L4) port number at the Internet Assigned Numbers Authority (IANA). Many well know examples exist, including port 23 for Telnet, 25 for Simple Mail Transfer Protocol (SMTP), 53 for Domain Name System (DNS) and 80 for Hypertext Transfer Protocol (HTTP). However, the following trends have led to a situation were L4 port number based application identification became highly unreliable.

- Many new applications simply do not register a port number at IANA, but rather they use either an already registered port number, a user-defined one, or a randomly selected.

- Numerous application designers intentionally use already registered port numbers to other well-known applications in order to hide their traffic thus avoid traffic filters and firewalls. A good example is Skype which uses port numbers 80 and 443 (originally assigned to HTTP and HTTPS by IANA).

- The exhaustion of the IPv4 address space caused extensive usage of Network Address Translation (NAT), where one public IP address is used by several endpoints having private IP addresses. For example, multiple physical servers may offer services through the same public IP address but using different L4 port number in order to avoid address collision.

In spite of these serious drawbacks, associating L4 port numbers with specific applications is still the fastest and simplest method for continuous monitoring and reporting, thus it is often used in practice when accuracy is not critical.

After recognizing the aforementioned phenomena, designers of traffic classification algorithms turned to a novel approach: instead of only looking at the structured information found in packet headers they started to examine the packets' payload content as well. These novel solutions match packet payloads against a so-called signature database which contains unique expressions for the different protocols in order to identify the application that generated the given traffic. This technique usually referred as Deep Packet Inspection (DPI). A good example is HTTP where the application header contains string literals for various information such as request

version (e.g. HTTP 1.1), request type (e.g. GET or POST), or user-agent (e.g. type and version of the web browser). Despite that DPI is considered to be the most reliable one among the traffic classification methods [57, 65] it faces three major challenges that limit it's capabilities:

- traffic encryption (e.g. communication over Secure Sockets Layer (SSL) or Transport Layer Security (TLS) channels) basically obfuscates packet payload contents, making it impossible to find protocol signatures inside them,
- well-know application signatures can be used as a diversion for hiding other data (e.g. injecting a dummy HTTP header before the real application payload), and
- pattern matching is a computationally expensive procedure which can limit the performance of DPI tools over high bandwidth links.

These concerns motivated other researches to examine further techniques suitable for traffic classification thus machine learning (ML) came into the picture. One of the most fundamental result in this field is [49] where authors showed that they can achieve relatively high classification accuracy by just observing of the first five packets of a TCP connection. However, the accuracy of ML based classifiers are highly dependent on following three elements [92]:

1. class of the ML algorithm (e.g. Bayesian techniques, decision trees, neural networks, etc.),
2. selection of classification features (most commonly used features include per-flow duration and volume, mean packet size, interarrival times of the first n packets),
3. reliability of the reference data used in case of supervised learning, also referred as ground truth.

This latter element is especially important: how can we even measure the accuracy of a traffic classification tool if we have an arbitrary input traffic stream from an unknown source thus we do not possess any reference data. Classification results by DPI algorithms are often used as a ground truth for testing ML based methods [111, 45], but for aforementioned reasons even DPI cannot guarantee 100% accuracy. To this

end, researchers presented traffic tagging tools that can be placed to the host machine of Internet users (e.g. a daemon application running in Windows) to generate ground truth data [45, 73]. However, since these works used human volunteers to generate traffic, sufficient data collection took a very long time (e.g. months). Moreover, in order to preserve the volunteers' privacy they are unable to share the traffic traces with other research groups, thus the results they present are not reproducible by others.

1.2 Research Objectives

The first goal of my PhD research (presented in Chapter 2) was to create a system capable of generating traffic traces used as an input for traffic classification tools. 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.

I created a novel framework called The User Behavior Based Traffic Emulator (UBE) [J1] which is able to generate traffic fulfilling these requirements. UBE 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.

In Chapter 3 I deal with the characterization of Internet users itself. In the past decade the research community gave a large attention to flow characterization. Flows has been classified in many ways, but most frequently by their size of traffic [110, 94, 68], by their duration in time [55], by their rate [81] and by their burstiness [81]. Several studies were written about the correlation between these flow behaviors [87, 91]. 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* [C6], 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*.

Moreover, based on these measurement result I gave a novel model where user bandwidth utilization can be provisioned by a modified Pareto distribution. 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 [85, 106]:

- 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.

Furthermore, based on the created model, I gave 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.

In Chapter 4 I investigate possible traffic measurement methods in Software Defined Networking (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 [88]. 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 [117, 79, 62, 112, 99], packet loss ratio [112], packet delay [112, 96], and route tracing [41]. 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. Hence in Chapter 4 I introduce a novel mechanism for measuring available bandwidth in SDN networks and validate the technique in Mininet emulation environment using multiple widely spred NOS platforms such as Floodlight, ONOS and OpenDaylight.

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

1.3 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. As I previously mentioned, accessing such high-speed traces is non-trivial if valid packet payloads are requirement, mainly due to user privacy concerns. The User Behavior Based Traffic Emulator (Chapter 2) 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 (Chapter 3) I analyzed these real measurements for creating bandwidth utilization model for Internet users. And finally, during my third thesis group (Chapter 4) I used the these traces to replay in emulated SDN network environment in order to evaluate the created available bandwidth measurement application.

Two of the measurement traces used my work 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 [5]:

- 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) [6] 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 1.1 I collected the statistics for all the three traces for further reference.

Table 1.1: Statistics of the three traces used throughout this my thesis.

	BME Wired Trace	BME WiFi Trace	CAIDA Trace
Number of packets	6804958	5796495	105444780
Number of users	1327	1970	680300
Number of flows	264117	307159	3876982
Total traffic	5.66 Gbyte	4.12 Gbyte	65.6 Gbyte

Chapter 2

The User Behavior Based Traffic Emulator

Internet Service Providers (ISP) are interested in the everchanging traffic characteristics of the Internet to develop efficient traffic management methods, charging policies, etc. It is of crucial importance that applications are accurately identified and their properties are clearly understood. In order to properly identify the application that generated a given flow, traffic classification tools are used. Deep Packet Inspection (DPI) is a subclass of traffic classification where the method relies on the inspection of packet payload content, instead of only looking at the structured information found in packet headers. Packet payloads are matched against a signature database which contains unique expressions for the different protocols. However, testing DPI tools in terms of both accuracy and performance is still an open issue in the research community.

Many recent publications compare the output of DPI tools in term of accuracy [111, 45, 108, 73]. The common method in these studies is the manual creation of ground-truth data by either one of the following two ways: i) run specific applications one at a time and filter out any background traffic that is unrelated to the given application [45], or ii) use a third party tool that can associate every generated packet to an application [111, 108, 73]. Although these methods proven to be an efficient way of testing the accuracy of DPI tools, the manual generation of test data is a highly unscalable process since it should be repeated frequently in order to test that the application-signature database of these tools are still up-to-date.

It is also well known that increasing accuracy by adding more and more signatures to the application-signature database negatively affects performance. The goal of the developers of DPI products is to provide high enough accuracy in real world networks with the highest performance. The most common solution for DPI performance testing is to use traffic simulators which mimic several application level network protocols (e.g., HTTP, SMTP), transport layer network protocols (e.g., TCP/IP, UDP/IP), and also user behavior (e.g., Poisson arrivals of user interaction events). However, simulators are not flexible enough by definition. They can simulate such traffic which is encoded in them. To create realistic traffic with a simulator, the simulator has to be also updated to keep up with the everchanging Internet [72]. This whole process is an overhead for the DPI signature set development which can be saved by collecting measurements with real protocol conversations in real network environment and replayed to the DPI box later. On the other hand, the network data is the property of the operator and plenty of privacy issues may arise if a DPI product vendor takes the measurements to its own site to further develop the DPI signature set.

The lifecycle of a DPI system comprises the steps shown in Figure 2.1. After testing a DPI device, the traffic falls into two categories: the recognized traffic part for which the system provides matching signatures and the unrecognized traffic part for which no signature provided any hit. Lack of continuous update of the signatures results in decreased number of signature hits and increased number of non-hits. This effect is due to the inevitable changes in existing protocols and the rise of new ones. Up-to-date active measurements containing latest traffic patterns are needed to update the signature set of the DPI box. The effects of the updates should be tested with traffic mixes containing hints for the new signatures while mimicking realistic network environment at the same time. During my research my goal was to create a system capable of generating traffic traces for testing purposes of traffic classification systems (especially for DPI tools) both in terms of accuracy and performance.

2.1 Other Traffic Generation Platforms

Numerous different traffic generators were proposed in the literature in the last two decades. In this section I mention several generally known solutions which are frequently referred in papers in the field of synthetic traffic generation.

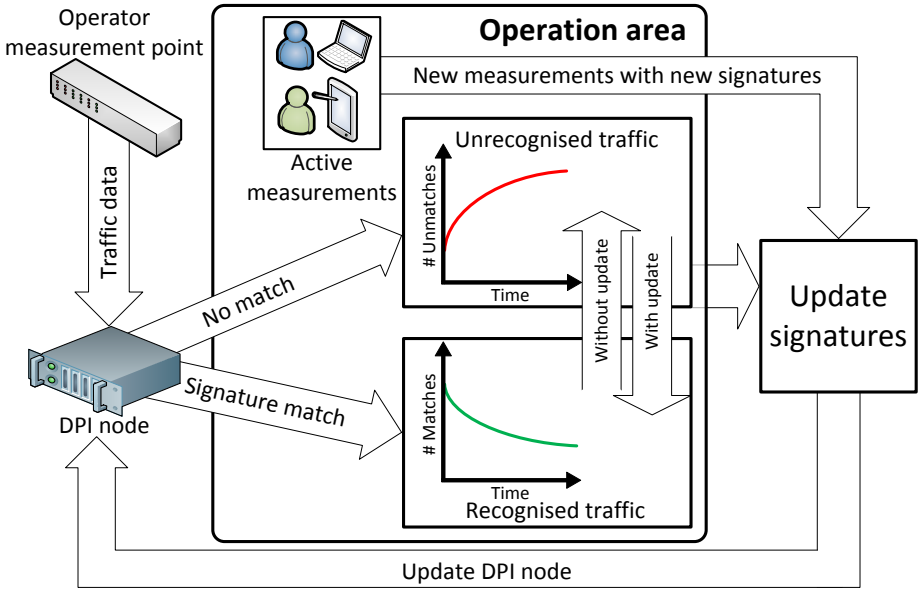


Figure 2.1: The lifecycle of DPI systems

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 [15] which can generate UDP packets at a given rate or TCP packets at maximum speed. BRUTE [47] 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 [46]. Other solutions, such as TG [33] or MGEN [17] supports different statistical distributions to be set up for the Inter Packet Times (IPT) and Packet Sizes (PS) information. Furthermore, Ostinato [27] is a very 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 generate 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 [32] which is a user-space application for replying *libpcap* files at arbitrary speeds onto the network. The software package also includes Tcplivereplay which is able to replay stored traffic using new TCP connections and by that adopting for the present network conditions. TCPivo [71] 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 [116] where authors replay OC-48 traces using multiple commodity PCs with Gigabit Ethernet network card. The collective drawback of these generators is that the measurement contains user sensitive information and cannot be distributed to other research groups for further work.

More sophisticated traffic generators are able to mimic the behavior of previously recorded traces by more complex traffic modeling. Harpoon [102] is a flow-based traffic generator that can mimic *netflow* based measurements by analyzing various flow characteristics. Swing [113] is a closed-loop, network responsive traffic generator which is able to extract distributions for user, application, and network behavior of real measurements. Tmix [114] 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 [52] is a comprehensive framework for synthetic workload generation. The tool supports 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 as in the previous cases: the model-based mode generates packet with synthetic payload and the trace-based mode arises privacy issues.

The idea of using GUI testing tools for controlling application in place of a human user was proposed in [115] where authors present a finite state machine model for driving applications. However, their automation only covers basic applications (e.g., Internet Explorer, Outlook and Microsoft Word) using an isolated testbed instead of the Internet and their goal is to present the effect of using anti-virus software on the system's performance. My goal is to provide repeatable traffic generation in more versatile environments including measurements with various access technologies and smartphone platform as well.

UBE does not belong to any of these categories since on the one hand, it captures the behavior of real users, and on the other hand, it generates output traffic streams

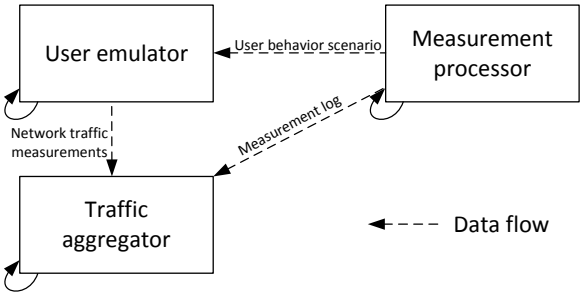


Figure 2.2: Abstract structure of the User Behavior Based Traffic Emulator

composed of traffic patterns taken from real measurements. It is also able to generate new user level measurements automatically ensuring that our database continuously contains the newest traffic patterns of different applications providing the ground truth data as well.

2.2 The Concept of the System

During the maintenance of a DPI box the protocol signature set has to be revised from time to time to check whether some of the signatures become completely obsolete or a new traffic type has emerged and the signature set has to be extended. The unrecognized traffic, i.e., the traffic which has no signature yet does not necessarily originate from a completely new application but a new version of an existing popular one extended with new features. The extension process of the signature set usually starts with active measurements. Selected applications are used one-by-one and regular expressions are constructed [109] on the recorded traffic. After software updates, the active measurements have to be redone. The measurements require the same user interactions with the application GUI from time to time. The basic idea of our system originates from the recognition that the manual repetitive work can be substituted with an automatic mechanism which is feasible due to th e practi ce that the GUI look and feel change less frequently than the underlying network protocol. A good example is Skype [48] which has the same skin from version 1-3 and it changed radically only in version 4. On the other hand, the underlying network protocol changed in several subversions.

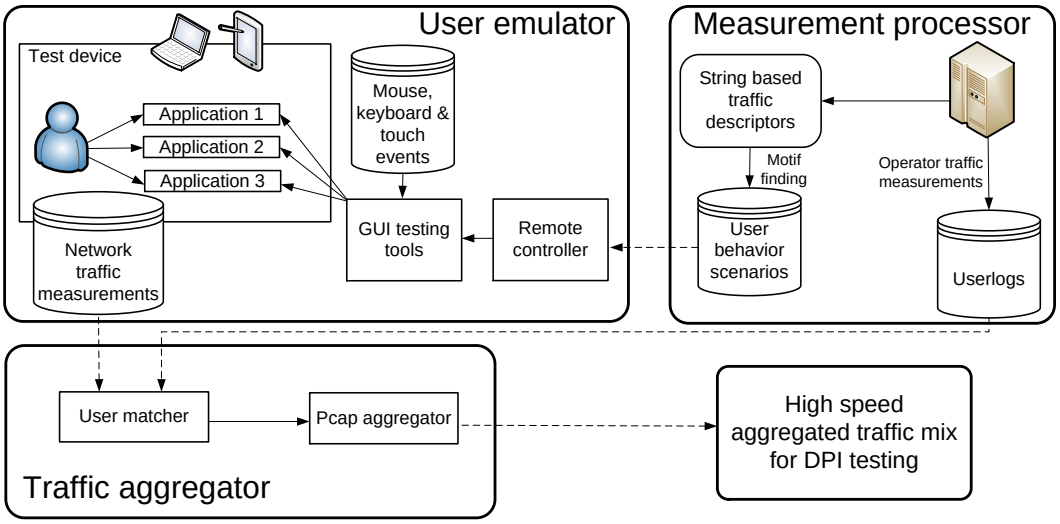


Figure 2.3: The architecture of the User Behavior Based Traffic Emulator

For terminology clarification the term *user behavior scenario* indicate a series of actions that a user does to interact with GUI applications. For example, the user opens a web browser, navigates to a torrent site, downloads a torrent file, opens it in a torrent application and five minutes later he or she closes it. Whereas, an *emulated user behavior scenario* means the process constructed by our framework in order to mimic a specific *user behavior scenario*.

In order to clearly present the architecture of the User Behavior Based Traffic Emulator (UBE) I present and discuss three figures in different levels of details:

- Figure 2.2: High-level abstract structure of the framework.
- Figure 2.3: Detailed functions of the framework.
- Figure 2.4: Data flow and database structure of the framework.

Figure 2.2 presents the abstract structure of the UBE. The framework is composed of following three main components. The *Measurement processor* is responsible for the definition of typical user behavior scenarios. The *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. The abstract structure shown in Figure 2.2 is detailed in Figure 2.3.

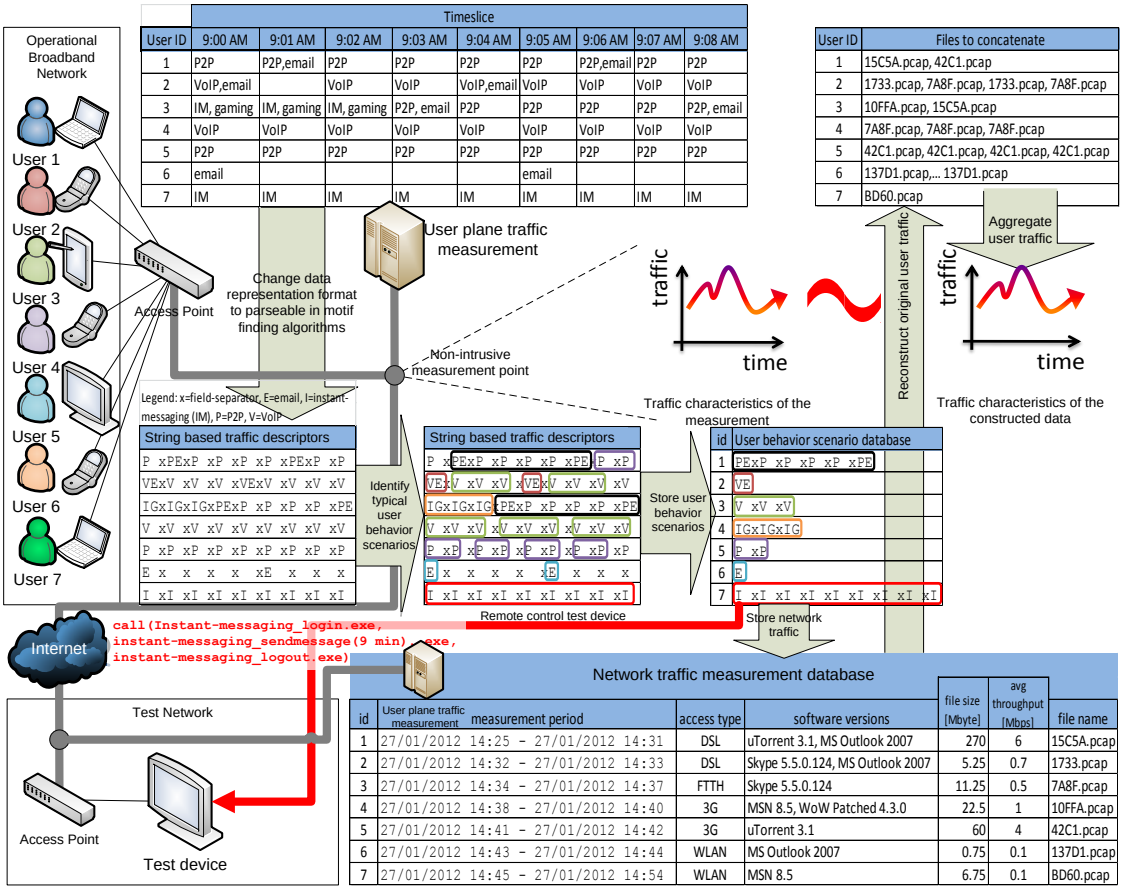


Figure 2.4: Data flows in the User Behavior Based Traffic Emulator

Although initial set up of the framework requires some operations detailed below, the main function of the three components in Figure 2.2 are repeatable and parallel. For further clarification, Figure 2.4 presents the data flow details and Figure 2.5 the time flow sequence diagram of operations, respectively. The detailed functions of the three main components of UBE are the following.

2.2.1 Measurement Processor

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.

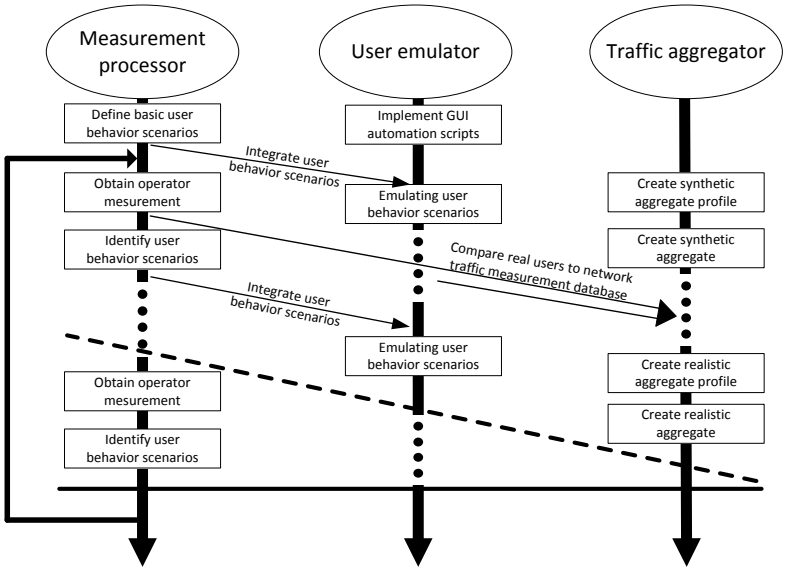


Figure 2.5: Time flow sequence diagram of the User Behavior Based Traffic Emulator

This process typically means the naming of the input fields, buttons, etc. – not the exact location of the mouse cursor – as Figure 2.6 shows. 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 (see Figure 2.3 and Figure 2.4 from *Operator traffic measurement* to *User behavior scenarios database*, for further details see Section 2.3.1). 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.

2.2.2 User Emulator

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*

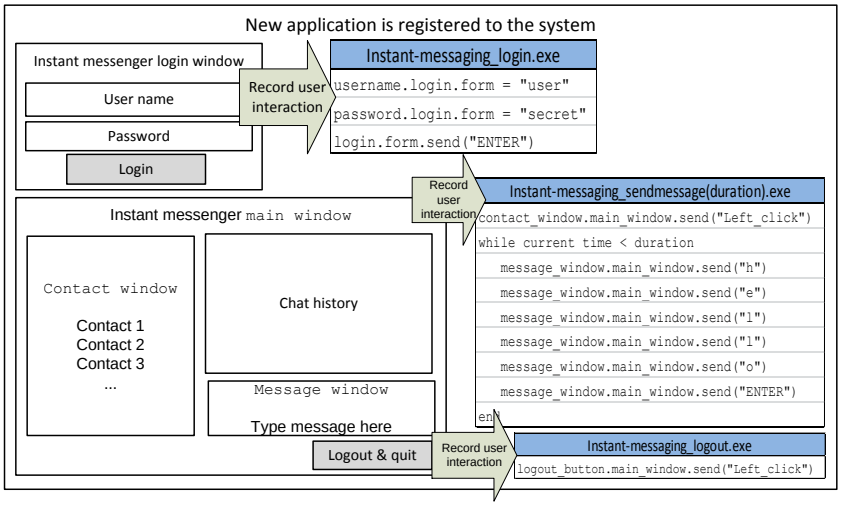


Figure 2.6: New application is registered to the framework

database (see Figure 2.3) is grabbed and user actions are emulated by remote controlling computers (see Section 2.3.3 for details) with the recorded user interactions (see Figure 2.6). During remote controlling (see Figure 2.3 *Remote controller*), 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 machi nes on d ifferent access network types for further increasing the diversity of the *Network traffic measurement database*. 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.

2.2.3 Traffic Aggregator

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* (see Figure 2.4 *Reconstruct original user traffic*). 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.

2.3 Methodology to Realize the Framework

In this section I give some insights on the implementation of the framework emphasizing on the elements interested in a researcher prospective. For complete implementation details see the help section of the portal [35]. Furthermore, the framework can be easily extended thus it is not limited to the applications or operation systems presented in this Thesis. Moreover, it is important to note that the framework had a successful demo presentation in [C1] as well.

2.3.1 Creation of User Behavior Scenarios

In the framework I defined the granularity of user behavior analysis in 1 minute scale. This means that one can say for each 1 minute period of the user what applications were used, e.g., only email occurred or also P2P file-sharing existed in the background (see Figure 2.4 *User plane traffic measurement*).

There are two possible ways to recreate the typical user behavior scenarios. One is a bottom-up approach, when one analyzes and stores small time slices of the user activity in our *user behavior scenario database*. Later when the original user activity has to be reconstructed there are a limited number of small building blocks in the database and, for example, a 100 min long user activity is constructed from 100 pieces of 1 min long slices. The other is a top-down approach where longest possible building blocks are matched one-by-one for the user activity with decreasing matching length, for example, a 100 min long user activity is constructed from 4 pieces of 20 min long slices, 1 piece of 10 min slice and 2 pieces of 5 min slices. Our goal was to focus on the top-down approach as it reduces the effect of transient states recorded during the

communication of applications. An example for such a transient state occurs during P2P file-sharing. At the beginning of the P2P file sharing session, when good seeders are searched for the network traffic has mainly signaling traffic exchange, while later this ratio turns in favor of the content exchange.

To extract typical user behavior scenarios the basic 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 [109]. 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. Our current goal is to select the smallest set of user behavior segments for the full coverage of the total user behavior sequences. To achieve this I constructed string literals per user from the packet-level network traffic measurement. In the *String based traffic descriptors* in Figure 2.4 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 [89] for further details on these string based traffic descriptors).

Currently there are 749 entries in the User Behavior Scenario Database which were created after analyzing multiple real measurements. The minimum and the maximum length of these scenarios are 4 and 10 minutes, respectively. Since the emulation is a real time process running all these scenarios takes about one week measurement on one test machine. Furthermore, using the current entries the available real measurements can be sufficiently covered.

2.3.2 Types of User Traffic

Two main types of generated traffic are identified in our framework. One requires the active attention of the user, thus the generating application is in the user's focus meaning that the specific application is the focused window. On the other hand, background activities are usually started once and later - after several other actions performed by the user - are switched off. The performed actions of the user behavior emulation consist of three main phases:

- *Starting phase:* This phase usually includes the starting of the application client or navigation to the starting page and login with user credentials. User credentials used by the framework were created solely for testing purposes. 3^{rd} party testers can change these information to their own and use them to build a database. However, these changes would not have significant impact on the payloads since credentials usually transferred via encrypted channels.

- *User activity or active phase:* In this phase some user actions are performed, e.g., sending some hotkey actions, mouse actions or other keyboard events.

- *Ending phase:* This phase is responsible of the proper logout and closing of the application.

The two activities are discussed as follows:

2.3.2.1 User Focus is Required

In this section I describe the implementation details of the user behavior emulation of those application types that require user focus.

- *Gaming*: I used World of Warcraft [39] to generate gaming traffic. The start phase opens the application and enters into the Public Test Realm (PTR) which is a special server used for testing the upcoming patches. Although PTR is not a regular server many players use it to test the upcoming changes in game mechanics. The active part of the emulation performs randomly some movements, spell usages or chatting. The ending phase closes the application. The framework is easily extendable with gaming scripts driving other popular online games and also compare the emulation results to relevant studies in this filed such as [64, 60].

- *Instant messaging*: I used Skype [48] for traffic generation (former version also included MSN Messenger [20] which has been integrated into Skype). The start phase opens the given application and performs the login of the user. The active phase picks a contact and starts sending messages to it. The messages are typed and sent with the exact timing I measured according to formerly

recorded chat logs. The emulation of typing is important due to the working mechanism of instant messaging applications which notifies the parties whether the other communicating party types or erases something. The ending phase logs out and closes the application.

- *Remote access*: During remote access emulation I used two kinds of popular applications: the built in Remote Desktop Connection [18] of MS Windows and RealVNC [29]. The starting phase establishes the communication tunnel. The user activity emulation phase performs some simple mouse and keyboard actions, while the ending phase terminates the connection.

- *Social-networking*: To be able to generate social-networking traffic I created a user on Facebook and 'liked' several pages to make the 'wall' full of new comments from time-to-time. The starting phase opens the website and navigates to random links inside the Facebook for the given time. I also switched of the caching function of the browser to download every data every time the script opens the same link. The script is also able to send messages to a randomly chosen friend according to the same log files I use during instant messaging.

- *Voice over IP*: Note that in some cases the synchronization of two clients is necessary. I needed two remote controlled computer for this type of activity and have Skype [48] installed on them. One of them is the call initiator, the other is the receiver. The call initiator picks the receiver computer user id from the contact list and performed a call with it. The application on the receiver side automatically accepts every incoming call. The script replays audio files containing human communication as input for both the caller and the receiver.

- *Web browsing*: To emulate web browsing activity a link is picked randomly from popular web pages [2] of the specific country the remote controlled computer is situated and a browser is opened with this URL. After loading the page, the active phase waits for a given time, browses the page for a while by rolling down on it and navigates to another either to a randomly chosen link on the current page or a randomly chosen URL from the original pool. The browsing phase can be important in case of AJAX [1] based dynamic web pages in which the

separate parts of the page are downloaded on demand, e.g., on the eBay [8] site.

2.3.2.2 Background Activities

In this section I enumerate the implementation details of the user behavior emulation of those applications which run in the background and do not require user focus.

- *File download*: To emulate file downloading traffic the starting phase begins file downloading by picking a random file from a formerly defined pool. After finishing, another one is picked and download is started. The ending phase stops the download and deletes every data from the download directory.

- *File sharing*: During file sharing emulation the file sharing client randomly opens a torrent file from a formerly defined pool. The pool contains torrent files in various sizes from different torrent sites. The pool also contains some magnet torrent files which don't use a centralized tracker server but rather other torrent hosts to find the given file to download (e.g., The Pirate Bay now shares only magnet links rather than regular torrent files). The ending phase deletes the downloaded data, thus reopening the torrent file results in restarting the whole file-transfer. The framework can emulate file sharing using uTorrent, Vuze and BitCommet.

- *Video playback*: Online video playback is either active being in the focus of the user, jumping in the video stream, clicking on new recommended videos, etc. or a completely background activity which plays all videos in a track list. UBE emulates the latter scenarios utilizing the channel function of YouTube. A playlist is loaded first and each of the videos are played one-by-one in the list.

- *Malicious traffic*: DPI devices could also be used for detecting malicious traffic. In order to further extend the functions of UBE I defined malicious traffic as a separate traffic type. I implemented two scripts which are able to download various malicious traffic in the background. The first script downloads the popular Eicar standard anti-virus test file [9] which is a harmless executable but most anti-virus product reacts as it were a virus. The second script uses the Malware Domain List database [16] to download a random malicious file.

Since this list is created for security experts, most of the links contains a real harmful program thus I only implemented it to virtual test machines where backup images are available.

2.3.3 Remote Controlling of the GUI on Desktop Windows Platform

For the emulation of user behavior I used AutoIt [4]. Its primarily goal is to make possible to create automation scripts or macros for Microsoft Windows programs. For every specific application client and for each phase (see Section 2.3.2) a specific script is constructed. The automation script can be compiled into a compressed and standalone executable file which can be run on computers that do not have the AutoIt interpreter installed. Moreover, AutoIt is compatible with every version of Windows from XP to 8.1 without recompiling the executable files. Also, previous implementations of the automation scripts used AutoHotkey [3] and Watir [38] but I found AutoIt the most suitable for the given task.

The standalone executable files has to be executed in a specific order according to the user behavior scenario that one intends to emulate (see Section 2.3.1). Applications with GUI, e.g., uTorrent [36] or Skype [48] have to be bounded to a graphical session in the Windows system otherwise running them directly from a console session would cause them unexpected errors. Thus the execution is performed from console but via an application called PsExec [28].

PsExec is invoked automatically from an external server by logging into the Windows machine via Telnet. Telnet session can be managed efficiently from the main server containing the user behavior database via Expect [11]. Expect is a simple script language created for automating interactive console based applications such as *telnet* or FTP.

2.3.4 Remote Controlling of the GUI on Android Platform

UBE is also able to emulate user behavior scenarios on Android platform using MonkeyRunner [19]. MonkeyRunner is part of the software development kit of Android and it is commonly used for stress testing applications as it can generate touch, drag

and keystroke events on the smartphones GUI. Although, MonkeyRunner only supports touch and drag events on exact pixels (rather than control buttons), by using the *intent mechanism* I was able to implement most of the emulation scripts for Android platform. *Intents* are abstract descriptions of an operation to be performed. It can be directly sent to an application or broadcasted into the Android system. In the latter case the *global intent filter* determines which application should get the message [14]. For example, sending an *intent message* with the Uniform Resource Identifier (URI) *www.google.com* will be directed to the default web browser or the URI *skype:testuser* will automatically open Skype and call the user named *testuser*.

The possibility to emulate users on multiple OS platforms using different access types with many applications gives us the opportunity to characterize our measurements in a similar way that was presented in [50]. For example, one could identify mobile users in the operational measurement and only compare them to dump files in our measurement database that was emulated on mobile platform. In [72] authors describe why it would be very complicated (if not impossible) to build a traffic simulator that can cover every possible network scenario. This is the reason why my approach needs a real measurement to mimic its behavior. This way an operator could contract traces similar to the conditions on his/her network. Using these mechanisms I also demonstrate the differences in the traffic pattern if the same user behavior scenario is emulated on different access and OS types in Sec. 2.4.

2.3.5 Recording of Network Traffic

For one specific usage scenario multiple measurements are created and stored in the *user behavior scenarios database* (see Figure 2.4 *Network traffic measurement database*) on the different test machines and setups. This is practically a link to a network measurement file recorded with *tcpdump* [31] during the emulation of the user behavior scenario. It is important to note that the Windows based traffic generating machines have a special driver (see [108]) installed to create dump files which can be perfectly classified later. This is achieved by a daemon which can track the opened sockets and modify the IP header according the application generated the current packet. Also, in Android platform I used similar approach that can track the opened socket and log it on the device (thus in this platform the framework does

not modify the IP packet headers). This tool is available on Google Play Store [37]. These mechanism provides the ground truth data on per packet granularity for every measurement in the *Network traffic measurement database* thus fulfilling this requirement against our framework. However, I also remark the tool which the framework using for ground-truth generation on Windows platform is not open for the public.

2.3.6 Deploying the System

In this subsection I give further insight into how one can deploy and use a similar framework. Figure 2.7 depicts the main components that have to be installed and also, the main outputs that the system generate. There are two components that need to be deployed: the control server and the test machines. The control server (in our framework a simple Linux machine) handles the databases (the *user behavior scenario database* and the *network traffic measurement database*) and controls the user emulation processes on the test machines. Deploying a test machine requires three steps. Firstly, every test machine has to be remotely controllable by the control server. For PCs it means an open telnet connection, whereas Android phones have to be attached to server via USB cable. Secondly, the GUI control scripts which are able to drive the applications have to be installed on the test machine . And finally,

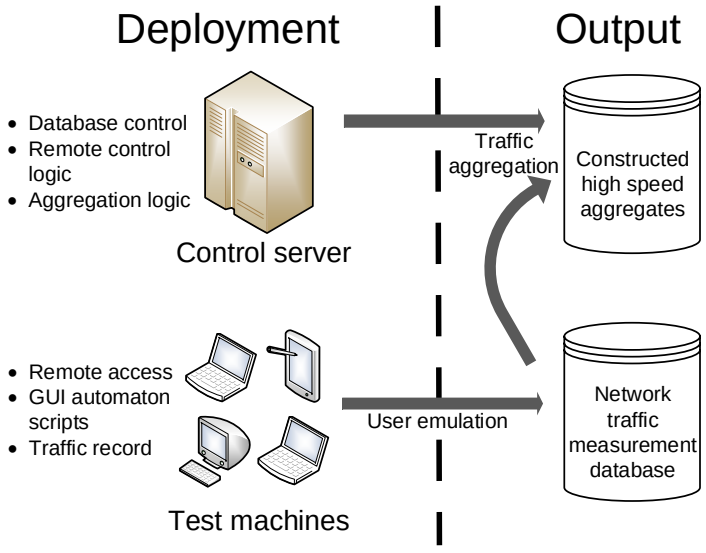


Figure 2.7: Main components to deploy the framework

test machines have to be able to record the network traffic in *libpcap* format. I used *tcpdump* on Linux based systems and *windump* on Windows system for this purpose. The logic of the *Traffic aggregator* is also stored in the control server. Most of the above mentioned scripts are available for download on UBE's website [35].

The user emulation processes generate the individual trace files in the *network traffic measurement database* which is one of the fundamental output of the system. The usage of these measurements is twofold. Firstly, by regular updates of these measurements the database can contain the newest application signatures thus it could serve as an input for automatic signature update tool (such system was presented in [109]). On the other hand, the *network traffic measurement database* is also the input for the *Traffic aggregator* part of the system where it is able to generate high speed aggregated traffic mixes which could be used in performance testing of DPI tools. An example for generating such high speed aggregate is presented in Section 2.5.

2.4 Application Studies

In this section I present analysis results of different traffic traces obtained by UBE. I do not intent to present every detail but rather I demonstrate the capabilities of UBE by some selected user scenarios which are present in both PC and smart phone platforms. Each presented scenario was emulated at least a hundred times on the test device of UBE using every possible access type.

2.4.1 Web Browsing

Recent studies showed that web traffic is dominant on smart phone platform [86, 69]. Towards understanding the differences between the same web browsing event in different circumstances I emulated the same scenario on every test device. Figure 2.8 plots the cumulative distribution function (CDF) of the packet inter arrival times (IAT) in downstream direction.

The CDFs related to the Windows platform and the Android using WiFi do not differ significantly. The reason of the difference is that desktop browsers download more data than smart phone's since many popular web sites have a mobile version which avoids using extra content e.g., flash based advertisements. However, using the

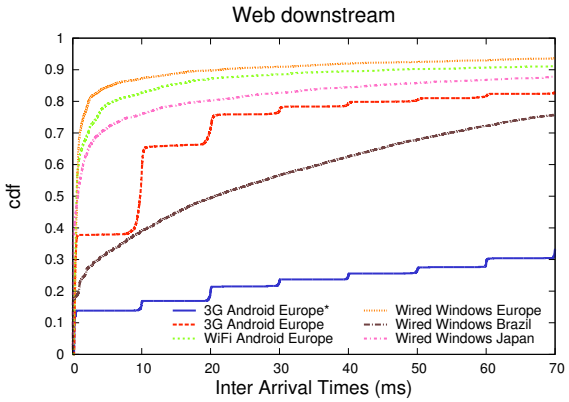


Figure 2.8: Web Browsing Measurement Results

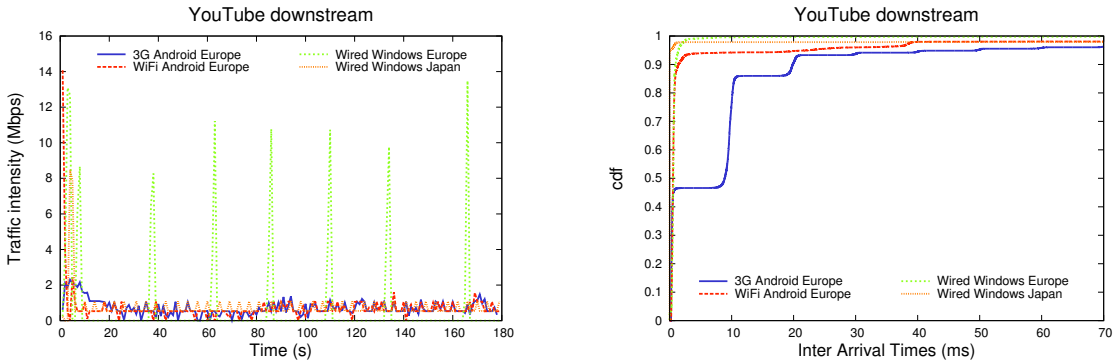
3G interface of the Android phones result in notably different values. In this case the IAT curve shows a tiered structure with 10 ms periodicity due to Node-B scheduling [77]. The deviation was even greater after the phone reached the monthly traffic limit and the operator limited its access speed to 128 kbps.

2.4.2 Media Streaming

According to [67] media streaming was 50 percent of total mobile traffic in 2016 and it's projected to increase 8-fold to 2022 accounting for more than 75% of world's mobile data traffic. YouTube is undoubtedly the most popular media streaming website being third in the world's traffic rank [2]. It's position should remain strong in the future since it's parent company, Google is the developer of the Android platform therefore the YouTube mobile application is installed by default on every smart phone with such operating system.

A recent study [44] has revealed that YouTube uses an additional application level flow control over the traditional TCP mechanism. At the beginning of the streaming an initial buffering period takes place where the server is sending data as fast as possible. This phase is followed by a block sending procedure of 64 KB sized blocks where the application reduces the sending rate close to playback speed. Figure 2.9a shows the traffic intensity measurement results which present the different cases that the YouTube application block sending mechanism can flow.

The results I have measured in our campus site using a desktop Windows shows periodic and very bursty traffic. YouTube flow control shows this type of pattern if



(a) Downstream Traffic Intensity (b) Downstream Inter Arrival Times

Figure 2.9: YouTube Measurement Results

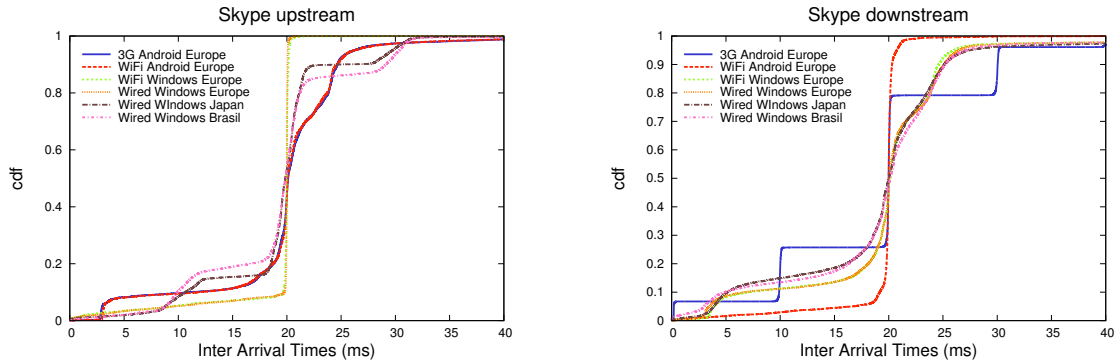
three conditions are fulfilled: (a) high speed access, (b) low round trip time and (c) no packet is lost due to congestion or buffer limit. Other Windows machines shows the normal flow of the 64 KB block sending period since a packet loss event occurred during the transmission. Android patterns show 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 [44].

Figure 2.9b plots the CDF of IAT of the YouTube flows. WiFi and wired measurements show the same characteristics regardless of the platform while 3G results have 10 ms periodic tiered structure due to Node-B scheduling [77].

2.4.3 Skype

Although VoIP applications do not share major portions of the total Internet traffic, they are very popular among smart phone users by the reason of free voice or video calls. On the other hand, mobile operators are working on identifying these kind of traffic for applying different charging polices than regular data service. Therefor understanding the traffic characteristics of VoIP applications is a crucial objective. In this subsection I present the analysis results using the most popular VoIP application, Skype. During every emulation process UBE has remotely controlled Skype the application used its default wideband codec: SVOPC [83].

Figure 2.10a presents the CDF curve of the inter departure times of consecutive Skype packets in upstream direction. I observed that the timing of these packets only depend on the platform and independent on the access type. In case of native



(a) Upstream Inter Arrival Times (b) Downstream Inter Arrival Times

Figure 2.10: Skype Measurement Results

Windows the timing is very precise to the 20 ms codec frame size, while on virtual Windows and Android platforms differ significantly from this value. The possible 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 [61]. Authors of [61] 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 [7] thus they don't have dir ect acce ss 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 2.10b 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.

Table 2.1: Traffic classification results for the *UBE database*

	Application	# Bytes	% Bytes	# Packets	% Packets	# Flows	% Flows
1	QuickTime	45 G	26.8 %	44 M	21.7 %	1016	0.03 %
2	Unknown	42 G	25.4 %	60 M	29.3 %	787 k	22.7 %
3	Flash	40 G	24.2 %	42 k	20.6 %	16 k	0.5 %
4	Bittorent	23 G	13.9 %	33 M	16.3 %	1.98 M	57.2 %
5	HTTP	13 G	7.8 %	19 M	9.1 %	376 k	10.8 %
6	SSL	525 M	0.3 %	830 k	0.4 %	26 k	0.75 %
7	DNS	55 M	0.03 %	340 k	0.16 %	163 k	4.7 %
8	Skype	50 M	0.03 %	220 k	0.1 %	22 k	0.6 %
9	Google	24 M	0.01 %	46 k	0.02 %	1170	0.03 %
10	ICMP	22 M	0.01 %	190 k	0.1 %	76 k	2.2 %
	SUM	167 G		206 M		3.5 M	

2.5 Validation Study

I carried out a performance evaluation study of the User Behavior Based Traffic Emulator to validate that the emulated traffic reflects similar characteristics compared to the traffic generated by users in real measurements. The validation of traffic generators can usually be performed from different points of views and on different time-scales [C4]. In this section 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.

The current database generated by UBE (all dump files in the *network traffic measurement database*, henceforth *UBE database*) contains about 1800 individual dump files, a total of 165 GB data, 200 millions of packets and 3.5 millions of flows. In order to investigate the traffic shares per applications in this database I classified

the traffic using nDPI which is an open source Deep Packet Inspection application developed by the nTOP project [21] and the results are presented in Table 2.1. Also, nDPI is considered to be one of the best performing DPI tools in the literature and it is also frequently upgraded by the developers [111, 45]. I used this methodology since later in this section I will show that the real measurement trace and the constructed trace by UBE generate similar amount of application signature matches using nDPI.

During this validation study I used the *BME WiFi trace* in order to showcase how UBE can mimic the statistics of a real broadband measurement. The trace is six minutes long and contains about 4 GB data and 5.5 million packets including the traffic aggregation of about 1970 users, 125k flows and 40 known applications.

I created the *constructed trace* via the *Traffic aggregator* component of UBE using

Table 2.2: Traffic classification results comparing the *BME WiFi* trace to the *constructed trace*

Application	Bytes		Packets		Flows	
	BME	Constr.	BME	Constr.	BME	Constr.
Unknown	2 G	1.95 G	2.9 M	2.8 M	36.3 k	43 k
HTTP	1.17 G	1.12 G	1.2 M	1.4 M	11.8 k	22 k
QuickTime	359 M	250 M	310 k	233 k	162	47
Bittorent	256 M	198 M	575 k	573 k	46 k	115 k
SSL	167 M	138 M	277 k	270 k	5186	9385
Google	42 M	3.7 M	64 k	9200	1025	95
Flash	23 M	94 M	24 k	97 k	97	431
DNS	7.7 M	4.9 M	42 k	27 k	20 k	13 k
Skype	1.9 M	4.1 M	15 k	39 k	1560	6600
ICMP	0.6 M	1.4 M	5441	11 k	2164	6170
SUM	4 G	3.77 G	5.45 M	5.45 M	125 k	217 k

Table 2.3: Statistical significance indicators of the traffic characteristic comparison of the *BME WiFi* trace to the *constructed trace*

Statistic	Mean		Standard deviation	
	BME	Constr.	BME	Constr.
Traffic intensity [M bps]	38.09	45.48	20.32	17.99 M
Packet size [byte]	771	680	693	668
Flow size [k byte]	17.4	10.2	905.9	766.2

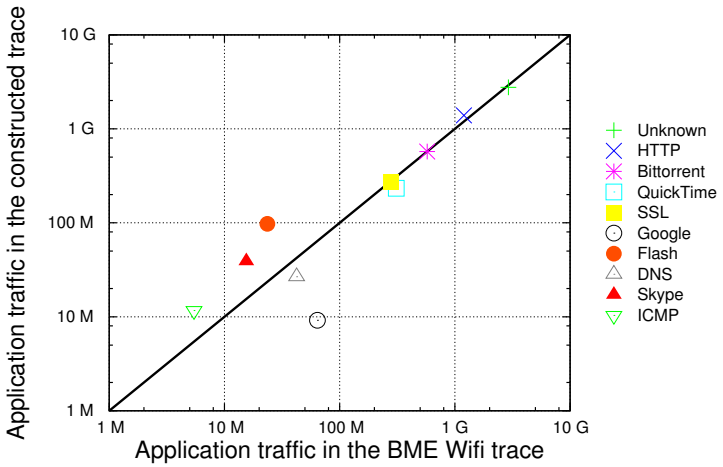
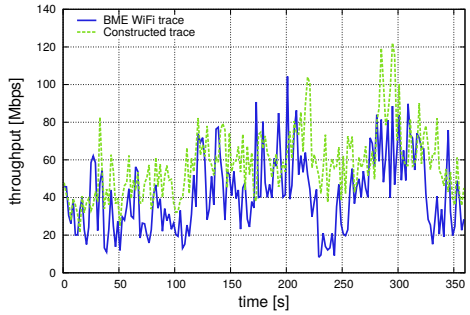


Figure 2.11: Application mix of the *BME WiFi* and *constructed traces*. The x-axis represents the traffic volumes of the top 10 applications in the *BME WiFi trace* whereas the y-axis represents the same traffic volumes in the *constructed trace*.

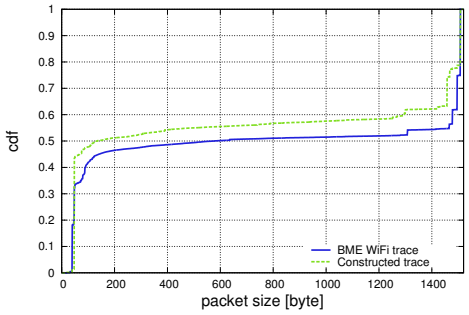
the available individual dump files in the *UBE database* as follows. Firstly, I considered the user level log from the *BME WiFi trace* used by the nDPI classificatory. This log contains the amount of data that were generated by every individual users in the aggregated measurement in a per application basis. After, UBE finds out which individual dump file from the *UBE database* is the most similar to a given user (see Figure 2.3 *User matcher*). The most similar measurement file is calculated by the following distance formula which can be considered as a metric in the application space: $\sqrt{\sum_i (O_{app_i} - P_{app_i})^2}$, where O_{app_i} and P_{app_i} is the amount of the i^{th} application data in bytes that were generated by the specific user in the operational measurement and in the specific dump file in our database, respectively. After this step, I had a list of dump files that should be concatenated to get a similar mix to the original operational measurement (see Figure 2.4 *Files to concatenate*). To get the final aggregated traffic I performed the reconstruction phase for every user existing in the trace. 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.

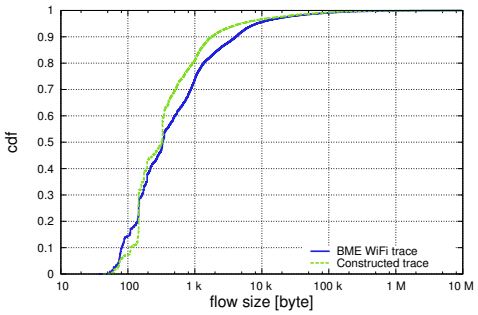
- Managing the IP addresses in the function of the number of emulated users. I had to alter the IP addresses of the test devices in the IP header. The framework



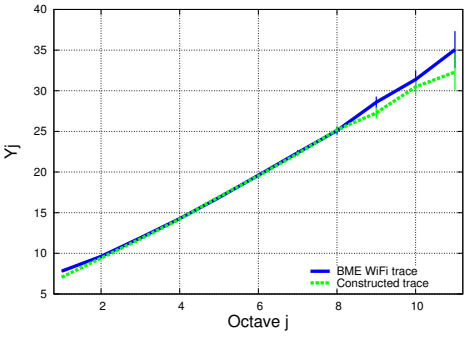
(a) Traffic intensity to downstream direction in the *BME WiFi* and *constructed traces*



(b) Packet size distributions of the *BME WiFi* and *constructed traces*



(c) Flow size distributions of the *BME WiFi* and *constructed traces*



(d) Logscale diagrams of the *BME WiFi* and *constructed traces*

Figure 2.12: Comparing the *BME WiFi* and *constructed traces* using different metrics

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.

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 and Table 2.2 presents the result of the top 10 applications after the classification for both the *constructed trace* and the *BME WiFi trace*. In addition, Figure 2.11 gives a general view about these 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 convention of nDPI.

It can be seen from Table 2.2 and in Figure 2.11 that the traffic shares of the top applications in the aggregation are correctly represented and important characteristics are also captured, e.g., Bittorent is the dominant protocol in terms of flows, a few

QuickTime flows cause reasonably large amount of traffic, or fairly large number of DNS and ICMP flows cause very small amount of traffic. On the other hand, traffic associated to Google web services is slightly over-represented in the *BME WiFi Trace* than in the constructed aggregate possibly due to the fact that many Android smartphone uses the campus WiFi network. However, this is a good indication that the *constructed trace* is suitable for performance 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. 3.2 and 3.4, respectively. Although, the throughput in the *BME WiFi* and *constructed traces* are not matching, the trends of the two curves in Fig. 3.2 show similar characteristics. This is further strengthen later in this section by a wavelet scaling analysis. In Figure 3.4 the Cumulative Distribution Function (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 of packets in the *constructed trace* (the total traffic of the *constructed trace* is about 5% less than the traffic in the *BME WiFi* trace, whereas the total number of packets is about the same in the two traces). Furthermore, in Table 2.3 I collected the mean and standard deviation values for the aforementioned curves. Although, the values have some deviation to each other, it can be considered close to approximate the characteristics of the original measurement.

To analyze flow-level characteristics of the two traces I plotted the flow size distributions. The CDF of the flow size is also well captured as depicted in Figure 2.12c. I observed an increased number of the torrent request-response activity (53 thousands of 145 byte flows) in the *constructed trace* compared to the *BME WiFi trace* resulting in a vertical jump in the CDF of the *constructed trace*. This is also the main cause for the slightly smaller values in the mean and standard deviation of flow sizes in the *constructed trace* presented in Table 2.3.

To investigate the scaling characteristic of the traffic I also calculated the logscale diagram [93] for both the *constructed trace* and the *BME WiFi trace*. The discrete wavelet transform represents a data series X of size n at a scaling level j by a set of wavelet coefficients $d_X(j,k),\ k = 1, 2, \ldots, n_j$, where $n_j = 2^{-j}n$. Define the q^{th} *order Logscale Diagram* (q-LD) by the log-linear graph of the estimated q^{th} moment $\mu_j(q) =$

$1/n_j \sum_{k=1}^{n_j} |d_X(j,k)|^q$ against the octave j. Linearity of the LDs at a different moment order q suggests the scaling property of the series, i.e., $\log_2 \mu_j(q) = j\alpha(q)+c_2(q)$, where $\alpha(q)$ is the *scaling exponent* and $c_2(q)$ is a constant. The plot of $\alpha(q)$ against q can reveal the type of scaling [93].

The scaling characteristics for both the *BME WiFi trace* and the *constructed traffic* are presented by the Logscale Diagram related to the moment order $q = 2$ in Figure 2.12d. 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 [93]. A linear regression to this interval gives an estimation of LRD parameter of $H_{BMEWiFitrace} = 0.875$ and $H_{constructedtrace} = 0.842$ 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 this validation study shows that the emulator is able to reproduce an aggregated traffic which captures the characteristics of the original measurements.

2.6 Application of Results

In this chapter I introduced the User Behavior Based Traffic Emulator (UBE), an automatic traffic emulation framework for constructing database for DPI testing. The system works by recording the traffic of remotely controlled computers and aggregating the traffic segments into multi-user traffic. UBE is able to construct a realistic aggregate traffic with arbitrary application mix, which is usually difficult to find in real measurements. Moreover, the generated traffic has no privacy restrictions so it can freely be distributed among DPI testing institutes. The emulated traffic contains up-to-date application level protocol information in the packet payloads and the characteristics of the traffic (e.g., application mix, packet sizes, flow sizes, scaling structure) exhibit the traffic characteristics measured in operational networks. The aggregated per user traffic was analyzed and validated by comparing several traffic characteristics with corresponding metrics investigated i n traffi c taken from real measurements.

A graduating student in our group used these traffic traces to evaluate five different traffic classification algorithms: a port-based classificator (based on IANA port

numbers [13]), SPID [10], TSTAT [34], OpenDPI [26] 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. [111] and [58]. 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.

Chapter 3

Pareto Characterization of Internet Users

In this Chapter I present my work on a novel characterization of user traffic. During the creation of the User Behavior Based Traffic emulator I used multiple real measurement traces to analyze the traffic generated by individual Internet users in order to create to most proper model. An interesting finding of my literature study was that flow characterization has a quite large attention in the scientific literature, whereas similar studies were not present regarding users. To that end, I conducted extensive user traffic analysis in similar manners. Based on the result I was able create a user model which approximates bandwidth utilization with a modified Pareto distribution. This model is unique in a sense that it takes into account the three main characteristics of Internet users [85, 106]: (i) the traffic proportion generated by individual users is very inhomogeneous such that a small number of heavy users generate the major part of the total traffic, (ii) users very rarely employ the offered maximal bandwidth especially in high speed optical access networks mainly due to the usage 802.11 devices and TCP limitations, and (iii) the average bandwidth of a single user over large time scales (e.g., in one month) is very low. Taking the results further, I also gave a formula that 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

3.1 Traffic Characterization Techniques

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*) [110, 94, 68], by their duration in time (as *tortoise and dragonfly*) [55], by their rate (as *cheetah and snail*) [81] and by their burstiness (as *porcupine and stingray*) [81]. Several studies were written about the correlation between these flow behaviors [87, 91].

There are several different definitions for *elephant flows* in the literature. In [94] 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 [68] 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 [81] become a rule of thumb in later literature (e.g. both [57] and [91] 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. [81] 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 [55]. Here, the authors considered a flow as *tortoise* simply if its duration was lager than 15 minutes. Given the generality and the rule of thumb nature of the definition by Lan and Heidemann [81] I will use the same definition for *elephants* later in my w ork.

In [101] 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 500ms 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 [84] 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 [97] 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.

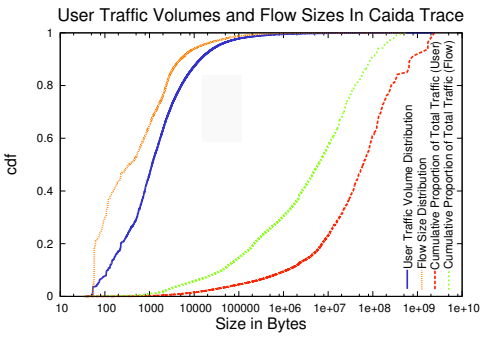
3.2 Elephant Users in Broadband Traffic

During the identification of *elephant users and flows* I use the definition presented in [81]: a user or a flow is considered an *elephant* if its flow size or traffic volume is grater than the average plus three times the standard deviation of all the flow sizes or traffic volumes of flows and users, respectively. In Table 3.1 collected the values of these threshold for the *CAIDA* and *BME Wired* traces. The *elephant and mice* phenomena clearly exist: less than a thousandth of the users and flows are responsible for roughly 60%-80% of the total traffic.

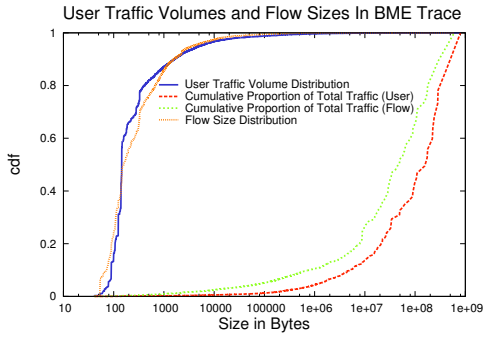
In this section I firstly show that the *elephant* phenomenon also exist with different threshold levels by plotting the cumulative distribution of user and flow sizes against theirs cumulative proportion of the total traffic. Furthermore, I present the comparison of the following three packet level metrics, (1) byte and packet throughput, (2) packet size distribution and (3) inter packet time distribution. I chose these metrics because they are the most frequently used packet level characteristics for comparing traffic traces [C4]. Additionally, I investigate presence of both *elephant*

Table 3.1: Statistics of the *CAIDA* and *BME Wired* traces.

	CAIDA Trace	BME Trace
Number of packets	105444780	6804958
Number of users	680300	63668
Number of flows	3876982	264117
Total traffic	65.6 Gbyte	5.66 Gbyte
Elephant user threshold	15.9 Mbyte	13.7 Mbyte
Number of elephant users	661	56
Proportion of elephant users	0.097%	0.088%
Total traffic of elephant users	71.5%	84.5%
Elephant flow threshold	2.3 Mbyte	4.96 Mbyte
Number of elephant flows	2714	151
Proportion of elephant flows	0.07%	0.057%
Total traffic of elephant flows	61.7%	83.41%



(a) CAIDA Trace



(b) BME Wired Trace

Figure 3.1: The *elephant and mice* phenomena presented by cumulative distribution of user traffic volume and flow sizes and their cumulative proportion of the total traffic

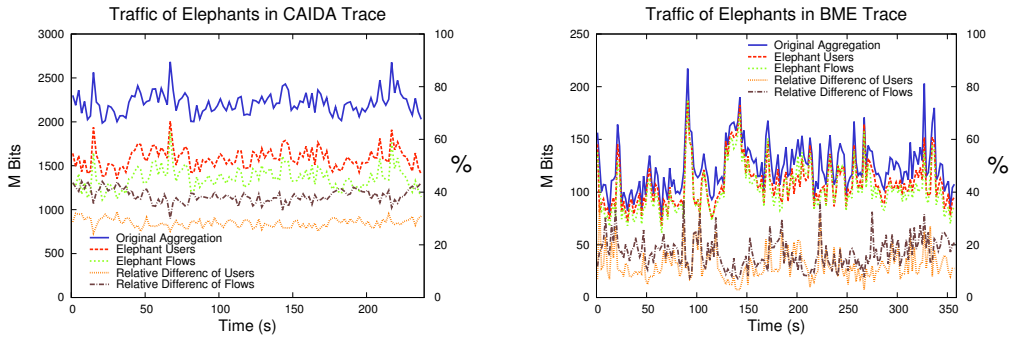
and *non-elephant flows* in the traffic of *elephant users*.

In Figure 3.1 one can investigate the *elephant and mice* phenomena for both traces. Here I plotted the cumulative distribution of user traffic volumes and flow sizes against theirs cumulative proportion of the total traffic. In Table 3.2 I collected the complementary values in percentage (1 minus the actual value) of the curves in Figure 3.1 for different thresholds. *Ratio* presents the proportion of users and flows whose traffic was larger than the *Threshold* value and *Traffic* represents their total share from the aggregated traffic.

Table 3.2: Proportion of *elephants* with different thresholds.

Threshold in Mbyte	CAIDA Trace				BME Wired Trace			
	Users		Flows		Users		Flows	
	Ratio	Traffic	Ratio	Traffic	Ratio	Traffic	Ratio	Traffic
0.1	2.17%	95.48%	1.21%	85.29%	1.21%	98.75%	0.53%	94.84%
0.5	0.8%	92.44%	0.29%	74.21%	0.53%	96.97%	0.16%	91.08%
1	0.54%	90.55%	0.18%	69.56%	0.37%	95.66%	0.11%	89.46%
2	0.37%	88.17%	0.11%	63.71%	0.23%	93.45%	0.08%	87.04%
5	0.23%	83.48%	0.04%	52.03%	0.14%	89.96%	0.06%	83.23%
10	0.13%	76.92%	0.02%	42.87%	0.1%	86.35%	0.03%	73.56%
15	0.1%	72.2%	0.017%	37.25%	0.08%	83.96%	0.025%	70.65%
20	0.08%	68.78%	0.011%	32.09%	0.06%	80.91%	0.02%	66.67%
50	0.03%	53.55%	0.003%	17.88%	0.03%	68.12%	0.007%	47.93%

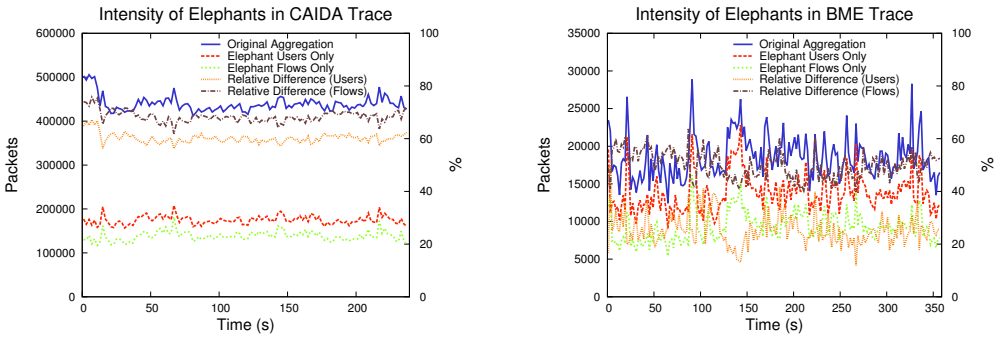
In Figure 3.2 the traffic of *elephant users* and *elephant flows* are plotted against the original traffic. The relative differences are also presented. In case of the *BME Trace* both the *elephant users* and *elephant flows* are responsible for sufficient amount of the total traffic (80%-85%), while in the *CAIDA Trace* this ration is a bit smaller (60%-70%). Since the traffic of both the *elephant users* and *elephant flows* seem to follow 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.



(a) CAIDA Trace (b) BME Wired Trace

Figure 3.2: Traffic of *elephants*

Figure 3.3 present the number of packets in every one second time interval. Here, the relative difference is much higher than in case of the byte throughput. In the *CAIDA Trace elephants* are responsible for only roughly 30%-40% of the total packets, while in case of the *BME Wired* trace this number is ratio is 50%-70%. Intensity of

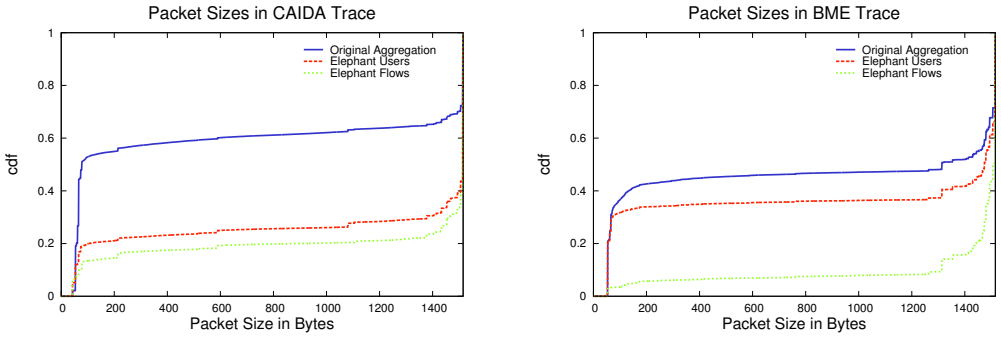


(a) CAIDA Trace (b) BME Wired Trace

Figure 3.3: Intensity of *elephants*

elephants are also following the packet burst of the original aggreagte since the relative difference is smaller in traffic peaks.

Packet size distributions of the two measurements is given in Figure 3.4. The joint property in both traces is that ratio of maximum and minimum sized packets is larger in *elephants* than in the original aggregate. Packets with intermediate size share similar proportion. I collected a few numerical example to Table 3.3 to present this phenomenon.



(a) CAIDA Trace (b) BME Wired Trace

Figure 3.4: Packet size distributions of *elephants*

Table 3.4 present the ratio of number of packet in *elephants* compared to the number in the original aggregate under different conditions. It is clear from the values that *elephants* contain the majority of maximum sized packet and *elephant flows* exclude the majority of minimum sized packets. The ratio of minimum sized packets in *elephant users* shows different behavoir in the two measurements.

Inter arrival time between consecutive packets corresponding the *elephant users or*

Table 3.3: Packet proportions under different conditions

Condition	CAIDA Trace			BME Wired Trace		
	Original Aggregate	Elephant Users	Elephant Flows	Original Aggregate	Elephant Users	Elephant Flows
PS <= 54 Byte	18.9%	11.9%	7.4%	20.9%	21.0%	2.9%
PS <= 66 Byte	44.3%	16.7%	9.9%	30.1%	29.4%	3.3%
PS >= 1450 Byte	32.8%	66.3%	72.8%	44.8%	54.6%	79.1%
PS = 1514 Byte	27.6%	54.2%	61.4%	28.5%	33.9%	49.7%

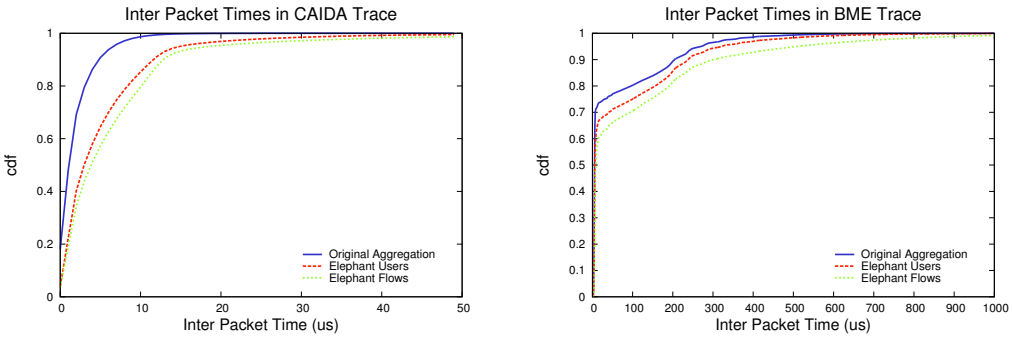
Table 3.4: Ratio of number of packets in *elephants* compared to the original

Condition	CAIDA Trace		BME Wired Trace	
	Elephant Users	Elephant Flows	Elephant Users	Elephant Flows
PS <= 54 Byte	25.0%	12.3%	74.7%	6.8%
PS <= 66 Byte	15.0%	7.0%	71.0%	5.3%
PS >= 1450 Byte	80.6%	70.2%	90.2%	88.6%
PS = 1514 Byte	81.2%	70.3%	88.4%	87.8%

flows are presented in Figure 3.5. 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.

In Figure 3.6 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 Wired* trace this number is only 3, but since there were only 56 *elephant users* in that measurement their share is 5%.

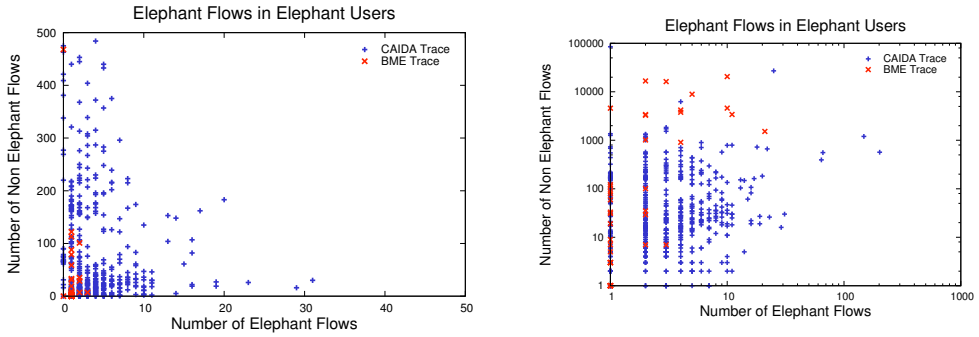
Another interesting result is that in case of the *CAIDA Trace* only the 9.13% of *elephant flows* were generated by *elephant users*. In case of the *BME Wired* trace this value is higher, namely 37.85%. These result clearly indicate that the overlap between the *elephant user* and *elephant flow* phenomenons could be much smaller in



(a) CAIDA Trace (b) BME Wired Trace

Figure 3.5: Inter Packet Time distributions of *elephants*

some cases that one would expect.

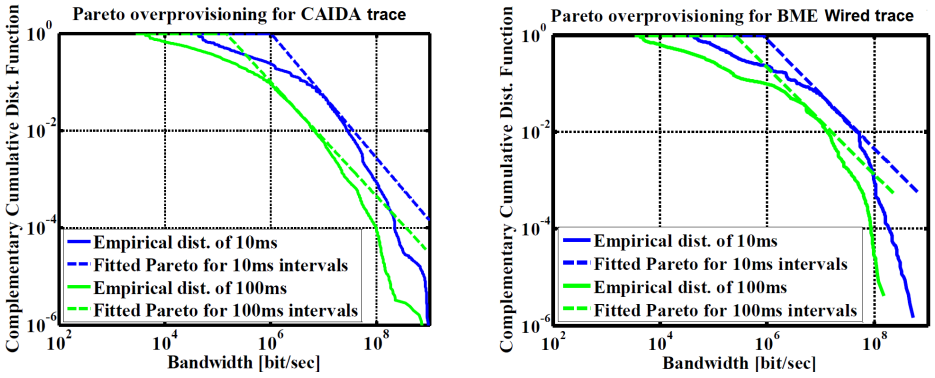


(a) Linear scale (b) Logarithmic scale

Figure 3.6: The number of *elephant* and *non-elephant flows* generated by *elephant users*

3.3 Novel Model for User Bandwidth Utilization

Based the results presented in the previous section, I created user model for user bandwith utilization. In this model I approximated users' bandwidth 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 3.7 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.



(a) *CAIDA* trace (b) *BME Wired* trace

Figure 3.7: Pareto distribution fitting for user bandwidth utilization in *BME Wired* and *CAIDA* traces

Table 3.5: Values after Pareto fitting in the *BME Wired* and the *CAIDA* traces

Window size	*BME Wired* trace		*CAIDA* trace	
	α	X_m	α	X_m
100 ms	0.90	250 kbps	0.85	144 kbps
10 ms	0.89	831 kbps	0.78	1016 kbps

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:

$$\widehat{F}_x(x, S_m, X_m, \alpha) = \begin{cases} 0 & \text{if } x < X_m \\ 1 - (\frac{X_m}{x})^\alpha & \text{if } X_m \leq x \leq S_m \\ 1 & \text{if } x > S_m \end{cases} \tag{3.1}$$

After fitting the Pareto distribution to the given measurements (shown in Figure 3.7) I collected the resulted α and X_m values in Table 3.5. The values also show close result for the *CAIDA* and the *BME Wired* 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.

The modification in Eq. 3.1 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:

$$M = \int_{X_m}^{S_m} f_x \, x \, \mathrm{d}x + \big(1 - \widehat{F}_x(S_m)\big) S_m = \frac{\alpha}{\alpha - 1} X_m - \frac{\alpha}{\alpha - 1} \frac{{X_m}^{\alpha}}{{S_m}^{\alpha - 1}} + \left(\frac{X_m}{S_m}\right)^{\alpha} S_m \quad (3.2)$$

Also, using the well-know 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:

$$\sigma^2 = \int_{X_m}^{S_m} f_x \, x^2 \, \mathrm{d}x + (1 - \widehat{F}_x(S_m)) {S_m}^2 - M^2 = \frac{\alpha}{\alpha - 2} {X_m}^2 - \frac{\alpha}{\alpha - 2} \frac{{X_m}^{\alpha}}{{S_m}^{\alpha - 2}} + \left(\frac{X_m}{S_m}\right)^{\alpha} {S_m}^2 - M^2 \quad (3.3)$$

For creating 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 ε that aggregated traffic of N users exceeds a given C_R limit. Usually, a service provider is interested in the capacity that should used for the aggregated link (C_R), for a given user number N and an availability rare ε. Thus I inverted the formula as follows:

$$C_R = N \, M + \sqrt{2 \, N} \sigma \, \mathrm{Erfc}_{inv}(2 \, \varepsilon) \quad (3.4)$$

3.4 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 provides.

In our work we were able to apply my model for such analysis [C7]. Using the formula given in Eq. 3.4 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, wheres above 600 Mbps they should deploy WDM based PON networks.

Chapter 4

Traffic Measurements in Software Defined Networks

In the past two decades, 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 [88].

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 [117, 79, 62, 112, 99], packet loss ratio [112], packet delay [112, 96], and route tracing [41]. 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.

In this chapter I present my work in the field of monitoring SDN networks which

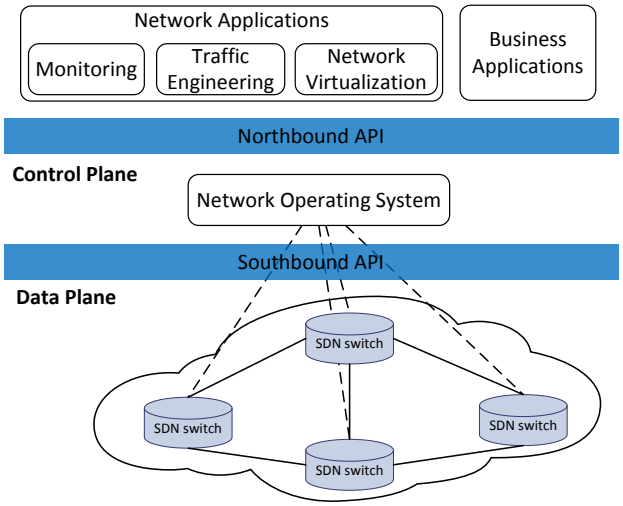


Figure 4.1: The architecture of Software Defined Networks.

is consists of the following three contributions. Firstly, I present the state-of-the-art Available Bandwidth (ABW) monitoring techniques used in Software Defined Networks emphasizing how they utilize the new features introduced by the architecture. Secondly, I 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, I present a novel method for available bandwidth monitoring in SDN and also validate experimentally in Mininet emulation testbed the analysis and the properties of the proposed technique. Moreover, based on my results together with my colleagues we proposed an extension to the OpenFlow protocol providing local timestamping mechanism in order to avoid measurement errors due to network jitter.

4.1 Background and Related Work

Software Defined Networking gained significant focus after the introduction of OpenFlow [88]. However, its main concepts root in earlier works in the fields of active networks, control and data plane separation, and network virtualization [70]. In my Thesis I follow the definition of SDN as presented in [80], 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. 4.1 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, but there are also other proposals, e.g. OVSDB [95], P4 [103] or ROFL [107]. For NOS platform there are many available open software such as NOX [22], POX [22], Floodlight [12] or Ryu [30]. Moreover, there are ongoing industrial consortia projects for controller platforms specialized for data centers, for e.g. OpenDayLight [25] or ONOS [24]. 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. The interested reader can refer to [80] for a comprehensive taxonomy of different elements in Software Defined Networking.

4.1.1 Traffic Monitoring in SDN

In the recent years, there has been several proposals for traffic monitoring Software Defined Networks. FlowSense authors [117] 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 [62]. Other papers propose to use the *FlowStatsReq* message in OpenFlow to poll the interface and flow counters in the switches for bandwidth measurement [62, 112, 99]. Furthermore, PayLess [62] and MonSamp [99] 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 [112] offers a solution for loss and delay monitoring as well. For loss measurement, it polls the flow counters on the ingress and egress switches for a given flow and calculates the difference. For delay measurement, it uses the SDN controller to inject probe packets into the network along a given path and than 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 [96] 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 I found that very little work has been published so far. One relevant work presents a variation on the Precise Time Protocol, named ReversePTP [90], aimed at distributing accurate time to SDN switches, allowing synchronized operations. An extension of the OpenFlow protocol has been proposed in [105] to add support for Synchronized Ethernet in SDN. Another approach focusing on delay is presented in [54], 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. The method that I proposed provides estimates not based on statistical model estimation and subsequent simulation, even though I 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 [54] the authors do not detail the error introduced by lack of time precision (possibly compensated for in the random sampling process).

4.1.2 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* [98].

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

$$\bar{u_i}(t-\tau, t) \equiv \frac{1}{\tau} \int_{t-\tau}^{t} u_i(x)dx \tag{4.1}$$

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

$$l_i(t-\tau, t) = C_i \cdot \tau \cdot \bar{u_i}(t-\tau, t) \tag{4.2}$$

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

$$\begin{aligned} a_i(t-\tau, t) &\equiv \frac{1}{\tau} \int_{t-\tau}^{t} C_i(1-u_i(x))dx \\ &= C_i(1-\bar{u_i}(t-\tau, t)) \\ &= C_i - \frac{l_i(t-\tau, t)}{\tau} \end{aligned} \tag{4.3}$$

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.

4.1.3 Available Bandwidth Measurement Methods

In traditional networks, available bandwidth estimation techniques are typically classified into active and passive. Active techniques send probe packets into the network and analyze how network traversal affected their spacing/arrival to infer network status. Most commonly used tools make inference about ABW through TCP/UDP achievable throughput. Tools like Iperf [15] or Ookla [23] use a single TCP/UDP flow to saturate the network path and estimate such achievable throughput. This technique is rarely used for continuous/frequent measurements in operational network since it severely interferes with existing traffic on the network. More sophisticated 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 [104] or Traceband [74] 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 [78] and PathChirp [100].

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.

A passive technique that leverages analysis methods developed initially for active ABW estimation is presented in [118], 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.

4.1.4 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 [53, 40]. 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 seconds up to 40 seconds, and—depending on configuration settings and traffic conditions—they may not converge to an estimation. The hybrid passive-active approach 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 led me to propose a novel ABW measurement technique in SDN as a complementary method with respect to active tools.

4.2 Measuring Available Bandwidth in SDN

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 I 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.

I proposed the use of a passive technique for the Available Bandwidth estimation, taking advantage of the NOS in the architecture of SDN. I used the northbound API to discover the topology of the network and to monitor the bandwidth utilization of the links. With this information the algorithm calculate the available bandwidth for any path in the network at any given time.

Table 4.1: 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	i^{th} link in the network topology graph
c_i	the capacity of e_i
b_i	the current bandwidth load on e_i
a_i	the available bandwidth on e_i, $a_i = c_i - b_i$
$P_{A \to B}$	the set of all available paths from A to B

Using the northbound API of the NOS the application queries the topology abstraction of the network which is a mandatory feature in every SDN controller [80]. 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. 4.1). 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 I used an approach similar to the one previously presented in [62, 112, 99]: the application periodically polls 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, the algorithm calculates the *available bandwidth* a_i on every link in the network, using (4.3). Based on the a_i values the application then calculate the available bandwidth on a given path P through the following equation

$$ABW_P = \min_{e_i \in P} a_i. \tag{4.4}$$

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. (4.4). 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:

$$ABW_{A\to B} = \max_{P\in P_{A\to B}} \min_{e_i\in P} a_i. \tag{4.5}$$

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. (4.4). 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 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).

4.3 Limitations and Constraints for Available Bandwidth Estimation in SDN

Based on the extensive analysis and measurements of the ABW over SDN networks, I have derived a number of limitations and constraints in estimating ABW with the technique I have proposed. These limitation also apply to every counter based measurement in SDN environment using the OpenFlow protocol. For each of them I provide an analytical modeling of the issue, an experimental evaluation in emulated environment, and possible solutions or mitigations.

4.3.1 Measurement Overhead

In traditional networks the measurement overhead caused by passive methods has been subject of several studies and proposals [62, 99]. Due to both the architecture of SDN networks and the different possibilities for monitoring it provides, measurement overhead can have multiple aspects. I report in Fig. 4.2 a visual breakdown of such aspects. Regarding *traffic*, measurement can affect the SDN control network (for passive

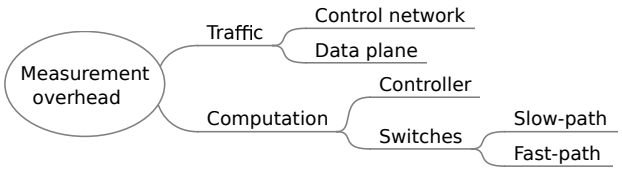


Figure 4.2: The impact of measurement in SDN networks.

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 then 1 Mbps.

4.3.2 Accuracy Limitation for Lack of Time 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 4.5.3. 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 δ_{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, and L_1 and L_2 are the counter values in the first and second measurements, respectively. The maximum delay in polling δ_{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 $C = C_{ctl}$, one can obtain

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

Given the size of *FlowStatsReq* message $Len_{query} = 56Bytes$[1], 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, a more frequent polling rate results in higher mesurement error.

[1] OpenFlow Switch Specification Version 1.5.1 Section 7.3.5.2

4.3.3 Critical Time-scale Dependence of Estimation

For its very definition, ABW is a metric that depends on a time interval (see Eq. 4.3), 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 seconds [76], while techniques provides a snapshot of the status of the whole path under measurement, averaging on sub-second time scale. However, 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.

4.3.4 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. 4.3 I show the message sequence chart depicting the communications between the ABW measuring application, the controller and the switches, during the measurement process; in the chart I named the time instants associated with notable events. With reference to Fig. 4.3, 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

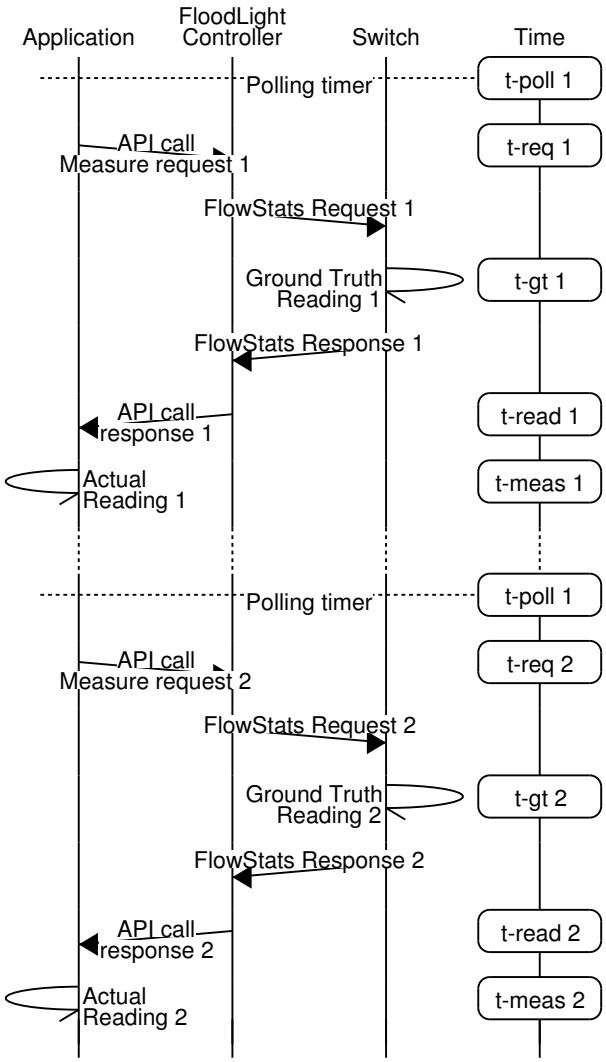


Figure 4.3: The measurement process.

t-read 1 and *t-meas 1*; the same applies to the each polling. I therefore approximated *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 message. By defining as ΔL the difference between the counter values L_1 and L_2 in the first and second measurements, respectively, the following formula can be obtained for the error:

$$\epsilon = \frac{\frac{\Delta L}{T^2_{GT}-T^1_{GT}} - \frac{\Delta L}{T^2_{meas}-T^1_{meas}}}{\frac{\Delta L}{T^2_{GT}-T^1_{GT}}} = \frac{\frac{1}{T^2_{GT}-T^1_{GT}} - \frac{1}{T^2_{meas}-T^1_{meas}}}{\frac{1}{T^2_{meas}-T^1_{meas}}} \tag{4.6}$$

In case I 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 i^{th} measurement period ($\delta_i = T^i_{meas} - T^i_{GT}$), Eq. 4.6 can be simplified as:

$$\epsilon = \frac{\tau}{\tau + \delta_2 - \delta_1} - 1 = -\frac{\delta_2 - \delta_1}{\tau + \delta_2 - \delta_1} \tag{4.7}$$

The formula in Eq. 4.7 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 were to model the network delay as $\delta = \delta_{min} + x$, where δ_{min} is the minimal possible delay between the switch and the ABW application and x is a positive random variable, δ_{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 I introduced a simple case where the network delay has normal distribution and analytically calculate the error. Based on Eq. 4.7 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 one can calculate the mean of the absolute error by the following.

$$
\begin{aligned}
E(|\epsilon|) &= \int_{-\infty}^{\infty} \left|\frac{y}{\tau - y}\right| f_y(y)dy \\
&= \int_{0}^{\infty} \frac{y}{\tau - y} f_y(y)dy + \int_{0}^{\infty} \frac{y}{\tau + y} f_y(y)dy \\
&= \int_{0}^{\infty} \frac{2\tau y}{\tau^2 - y^2} f_y(y)dy \\
&\overset{\tau \gg \sigma}{=} \int_{0}^{\infty} \frac{2y}{\tau} f_y(y)dy = \sqrt{\frac{4}{\pi}} \frac{\sigma}{\tau}
\end{aligned} \tag{4.8}
$$

Eq. 4.8 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 4.5.3.

I explicitly notice that, even if affected by said sources of error, *ABW estimations provided by the presented 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, my technique presents several advantages and enables new applications of ABW, not previously considered in the traditional scenario: I discuss such advantages and enabled applications in the following section.

4.4 Advantages and Novel Applications of Available Bandwidth Estimation in SDN

The newly presented technique for estimating ABW in SDN 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 for a time granularity and estimation accuracy of orders of magnitude better than the ones reached by active techniques in traditional scenarios (see Section 4.5.3 for more details), making several unprecedented applications possible.

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

- Highly-dynamic routing [43, 63].
- 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 [56].
- 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 [42].
- 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 [59].

4.5 Experiments over Mininet Testbed

In order to evaluate the available bandwidth measurement application presented in Section 4.2 and also, to validate my statements in Section 4.3, I conducted extensive network emulation scenarios using Mininet [82]. Fig. 4.4 presents the schematics of our testbed. During the emulation scenarios SDN switches are represented as running

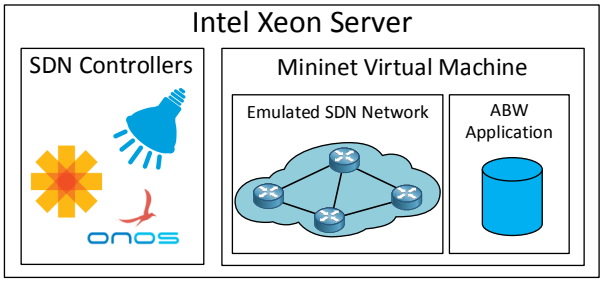


Figure 4.4: The assembled test configuration.

Table 4.2: 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[1]	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

Open vSwitch entries and they can be connected to three different SDN controllers. Mininet emulation runs the switches inside a separate virtual machine[1] 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 [75]. However, I chose to run the controllers directly in the host server since I often faced load issues when I run it inside the virtual machine. For the proof-of-concept application in Sec. 4.5.2 and 4.5.3 I used Floodlight [12] as NOS since it is easy to use and provides the exact features I needed to validate the theoretical results. In Sec. 4.5.4 I created the same application for industrial controller platforms OpenDaylight [25] and ONOS [24] and I found fundamental differences on how they collect the statistics from the SDN switches. For further reference, I collected the

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

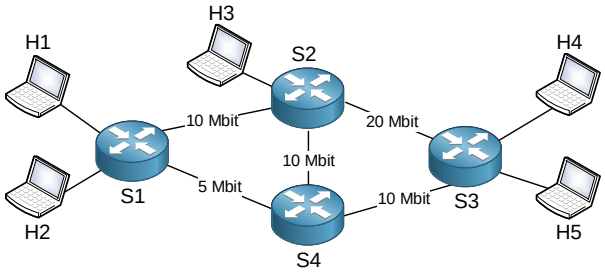


Figure 4.5: The test topology in Mininet.

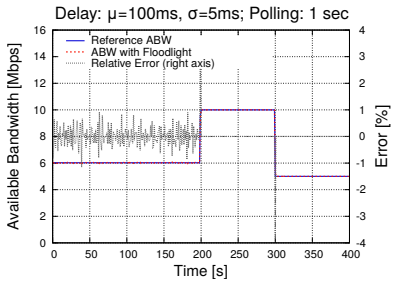
used hardware and software versions in Tab. 4.2.

During the emulation scenarios concurrently with the measurement application, I 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. I used 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.

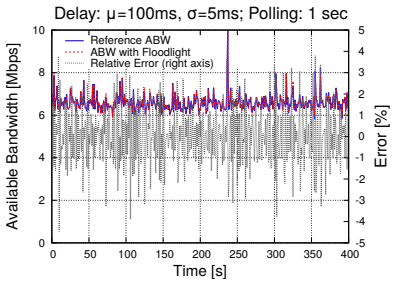
4.5.1 Test Configuration

Fig. 4.5 sketches the network topology I created in Mininet for testing purposes. S1, S2, and S3 create a classical Y topology which is frequently used as a testbed for testing ABW applications [53]. My idea was 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 [53]. To realize such scenario, I used *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.

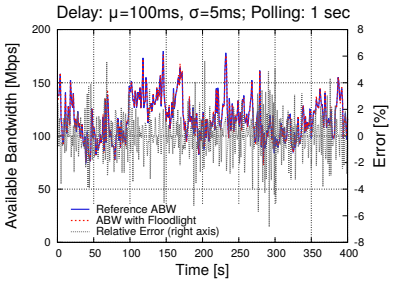
I avoided sending any traffic through S4, as the default route policy in all the three controllers do. This was to use the feature in our application which can predict the best possible alternative route even if such route is not the default one. If the volume of cross traffic from H3 to H4 is larger than 10 Mbps, the alternative route through S4 would provide path with larger available bandwidth.



(a) CBR traffic measurement.



(b) VBR traffic measurement.



(c) Real traffic measurement.

Figure 4.6: Measuring bandwidth on the link between S2 and S3 and the available bandwidth over the Mininet test network.

4.5.2 Validation of Available Bandwidth Application

During the validation process I used D-ITG [52, 66] for traffic generation, since it was proven to work much reliably than other traffic generation platforms [51]. Using D-ITG, I defined the following three traffic scenarios in order validate the our ABW application in different circumstances:

1. *Constant Bitrate (CBR) Traffic.* In this scenario I generated three flows with constant bit rate with the following timing. At the beginning of the measurement, H1 starts to send 4 Mbps of UDP traffic to H5 for 100 seconds, than the host sleeps for 100 s (generating no traffic) and restarts sending with 8 Mbps rate. Parallel to this, H3 starts to send 10 Mbps of UDP traffic to H4 for 100 s after the start of the measurement until the end. Fig. 4.6a presents the traffic from H1 to H5 and from H3 to H4, and also the available bandwidth between H1 and H5. Although the bandwidth measured by the Floodlight controller is varying due to the variable 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 path is S1→S4→S4 with 5 Mbps available bandwidth.

2. *Variable Bitrate (VBR) Traffic* is generated by D-ITG using Pareto distribution for the inter-departure times of packets. I generated two flows, one from H1 to H5 and the other one from H3 to H4. I tested different values of shape and scale parameters and report the most interesting cases in the following. Fig. 4.6b plots the traffic from H1 to H5 and from H3 to H4, and the available bandwidth between H1 to H5. In this case I used $\lambda = 1.75$ as shape parameter for both flows, whereas for scale parameter I used $X_{H1} = 1ms$ and $X_{H3} = 0.5ms$ for H1 and H3, respectively. As shown, the error of the ABW measurement is larger than in case of CBR traffic using frequent polling rates. On the other hand, since the inter-departure times of packets are identically and independently distributed on larger time scales, the traffic becomes smoother making the error rate similar to CBR results.

3. *Real Traffic.* For realistic traffic generation, in this cased I used the *BME WiFi* trace to replay. I 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 I used for the previous cases, I also increased the capacity of the links tenfold. Fig. 4.6c presents generated traffic and the ABW measurement in this scenario. In this case the throughput also varies in larger time scales, thus one can 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. 4.6, 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 presented in Section 4.3.4.

Table 4.3: Error rate of ABW measurements using real traffic replayed by D-ITG with no added delay.

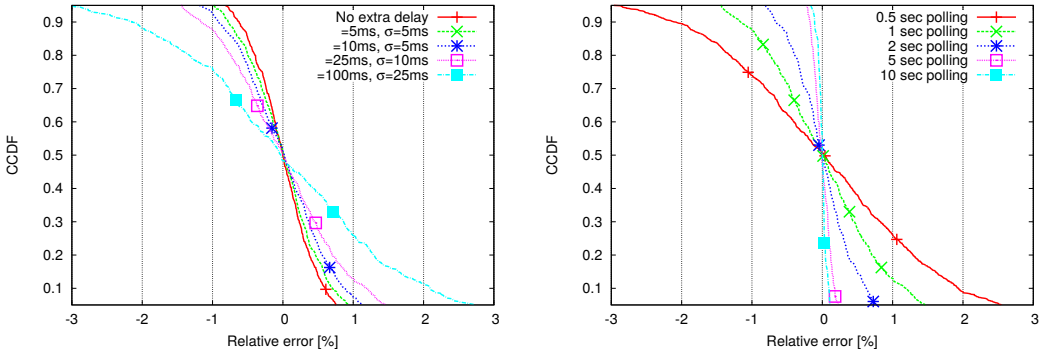
Error [%]	Polling period in seconds								
	0.5	1	2	5	10	20	30	40	60
Mean	0.72	0.39	0.18	0.085	0.055	0.021	0.013	0.010	0.0065
STD	0.94	0.51	0.23	0.010	0.074	0.024	0.016	0.013	0.0069

4.5.3 Analysis of Measurement Error

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 no delay extra delay on the which can be considered as an ideal case. Table 4.3 reports the mean and the standard deviation of the measured error values for such cases. However, it can 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 I average on a larger time scale while the difference in the timestamp approximation remains the same, thus having a smaller relative effect. Note that I always used 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 I found that 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 [40], the average error rate is less than 10^{-5} which can be considered as very accurate.

Hereafter, I present the cases were I introduced artificial delay between the SDN switches and the Floodlight controller to investigate its effect over the error rate. Fig. 4.7 presents the CCDF of the error for different delay values using frequent polling rates. In Fig. 4.7a I fixed the polling period to 1 sec and used different delay values between the switches and the Floodlight controller. 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, respectively.

In Fig. 4.7b I show only one delay value (25 ms mean with 5 ms standard deviation)

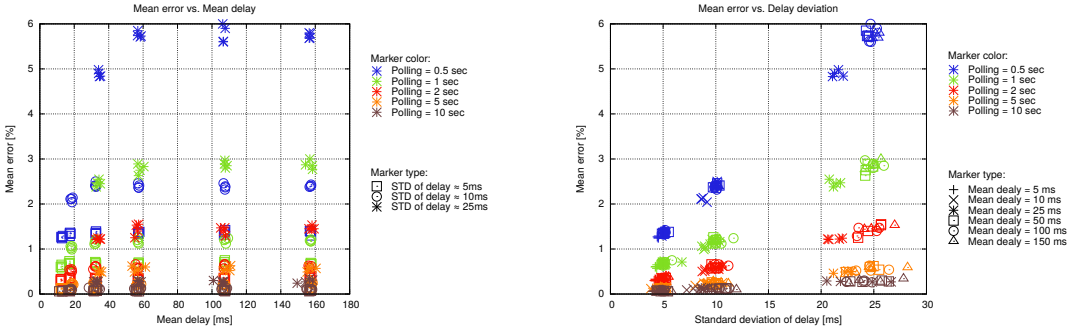


(a) Error values for 1 ms polling using different delay.

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

Figure 4.7: 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).

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 my previous calculations 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 one can achieve more precise ABW values in case very frequent results are not needed. This leads to the conclusion that the proper value for polling rate and maximum network jitter acceptable is a function of the application



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

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

Figure 4.8: The average error measured during real traffic replay using different mean delay and standard deviation setups.

Table 4.4: Error rate of ABW measurements using real traffic replayed by D-ITG.

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

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. 4.8 I plotted the result of every measurement with real traffic. I 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. 4.8a and 4.8b, respectively. The results in Fig. 4.8a confirms my statement in Sec. 4.3.4 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. 4.8b 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).

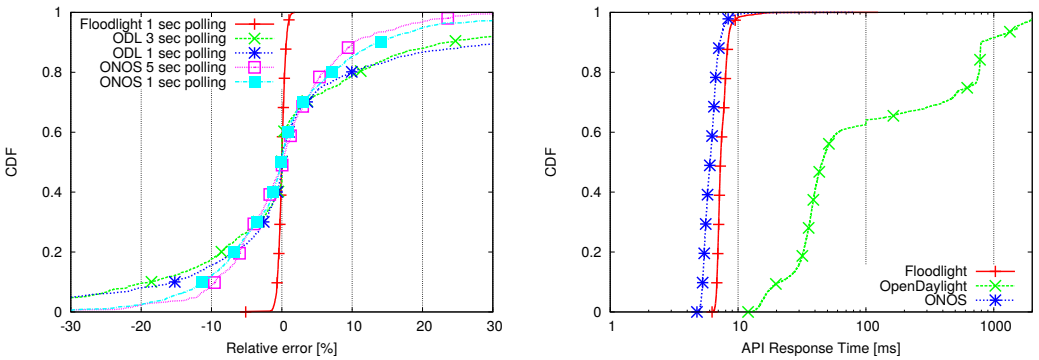
Tab. 4.4 summarizes the mean and the standard deviation of the measurement result in the most interesting cases. These results clearly show the trade-off constrains between the error rate, the polling frequency, and the monitoring packet delay. Applications working with SDN networks and in need of ABW estimations can be properly devised looking at these results.

4.5.4 Measurements with Industrial Controller Platforms

In order to test out application in a wider spectrum I also prepared the same ABW application over OpenDaylight (ODL) [25] and ONOS [24] which are the two most

commonly used SDN controllers in industrial environment. Since both of these controllers provide an API access for querying the switch port counters I 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 I assumed for the ABW application and what I presented in Fig. 4.3. When the application initiates 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 a possibility to modify this rate. However, both ODL and ONOS do not provide a timestamp value for the measured port counter values when someone queries them via the APIs. This phenomenon made the application to estimate the available bandwidth very poorly since the measurements (with the reference value generation) were out of phase from the collection of the controllers. Fig. 4.9a 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. I also conducted measurements when the polling rate in the ABW 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 I suggested 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. Using such timestamping one could 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 I measured almost two orders of magnitude larger response times than in case of Floodlight and ONOS. I report the CDF of the API response times for the three considered controllers in Fig. 4.9b. Since ODL is very complex software platform I did not attempt to find the exact 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



(a) Error value comparison of different controllers.

(b) API response time comparison of different controllers.

Figure 4.9: The CDF of ABW measurement error and API response time generated by our application over OpenDaylight, ONOS and Floodlight.

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.

4.6 Application of Results

While the access to switches counters highly simplifies most monitoring tasks, I have theoretically (Sec. 4.3.4) and experimentally (Sec. 4.5.3) 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 uncertainty in the estimation of ABW, therefore together with my colleagues we advocated 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[2], 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). For the message formats we proposed different implementations, described hereafter.

4.6.1 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 Section 7.5.5" 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. 4.10 for the format of the proposed implementation for the generic Experimenter message with the addition of a Timestamp).

4.6.2 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

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

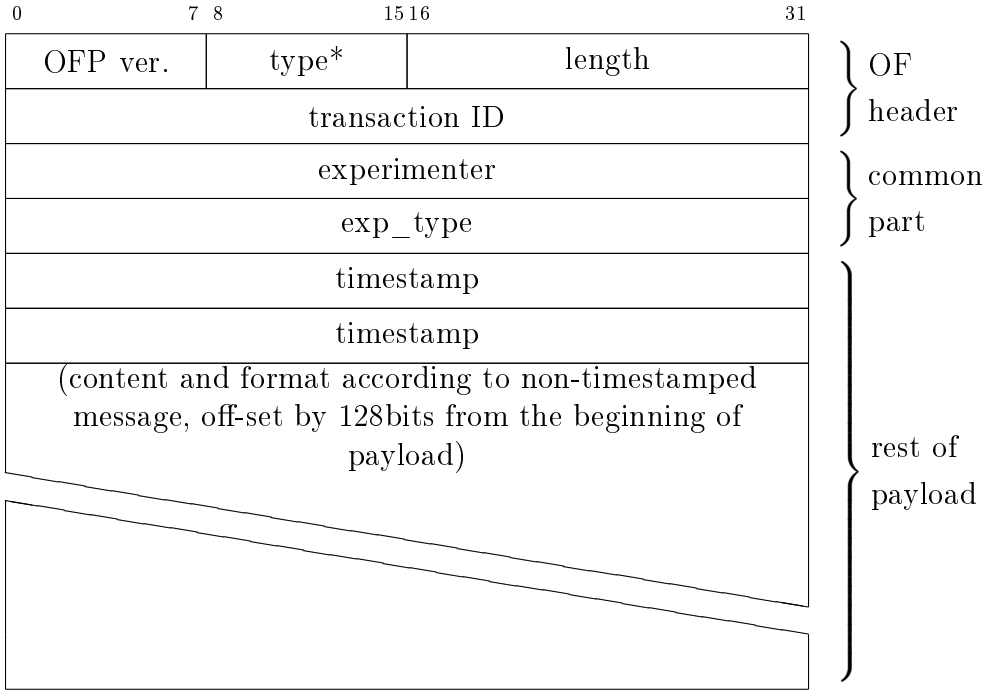


Figure 4.10: Format of OpenFlow Experimenter message with Timestamp.
* *type* header field is set to value *OFPT_EXPERIMENTER*.

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 proposed 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 [3]. We proposed 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.

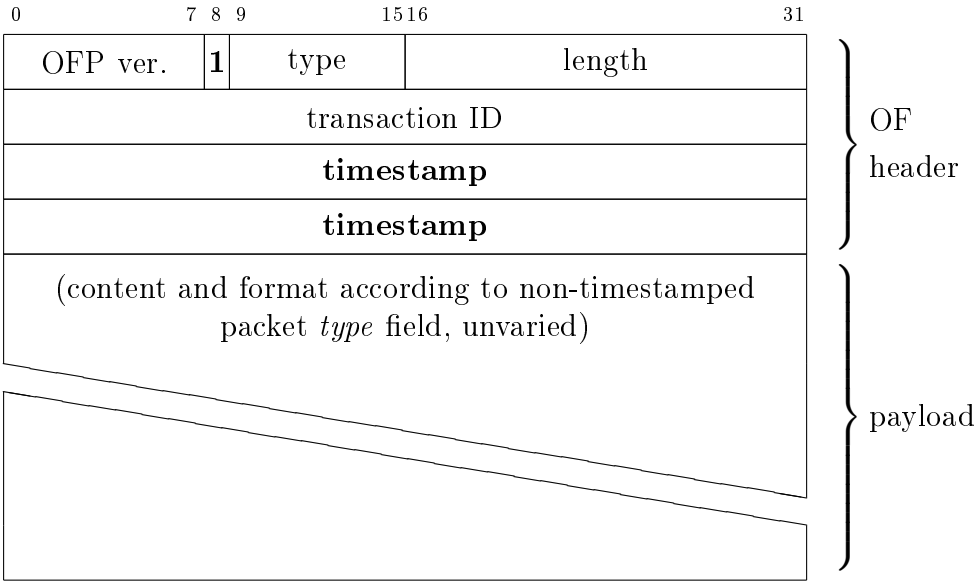


Figure 4.11: Format of OpenFlow reply packet with *Timestamp Flag* set. Changes against OFv1.5.1 are highlighted in bold.

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.4.11). 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 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*

[3] OpenFlow Switch Specification Version 1.5.1 Section 7.1.1

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.

Chapter 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 (UBE) which is able to generate input for traffic classification tools. The system works by recording the traffic of remotely controlled computers and aggregating the traffic segments into multi-user traffic. UBE is able to construct a realistic aggregate traffic with arbitrary application mix, which is usually difficult to find in real measurements. Moreover, the generated traffic has no privacy restrictions thus it can freely be distributed among DPI testing institutes. The emulated traffic contains up-to-date application level protocol information in the packet payloads and the characteristics of the traffic (e.g., application mix, packet sizes, flow sizes, scaling structure) exhibit the traffic characteristics measured in operational networks. The aggregated per user traffic was analyzed and validated by comparing several traffic characteristics with corresponding metrics investigated in traffic taken from real measurements. Therefore the presented framework is quite unique on the palette of numerous traffic generation tools. I summarized my result regarding the framework in Chapter 2.

In Capter 3, I continued my research with the characterization of user behavior by creating 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. In Chapter 4, I presented the state-of-the-art techniques for this emerging architecture, emphasizing how they utilize the new features available and what are their limitations. I 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. My results show that my 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. I also implemented the ABW application over the two most commonly used industrial SDN controllers: OpenDaylight and ONOS. I 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 I suggested to extend the API and the statistics collector modules of both ODL and ONOS with a timestamp providing mechanism. Finally, based on my result together with my colleagues we 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, whereas the other uses 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.

Bibliography

[1] AJAX tutorial. http://www.w3schools.com/ajax/default.asp, retrieved: Sept., 2017.

[2] Alexa: Top 500 Global Sites. http://www.alexa.com/topsites, retrieved: Sept., 2017.

[3] AutoHotkey. http://www.autohotkey.com/, retrieved: Sept., 2017.

[4] AutoIt. http://www.autoitscript.com/site/autoit/, retrieved: Sept., 2017.

[5] BME Traffic Traces. http://megyesi.tmit.bme.hu/traces/, retrieved: Sept., 2017.

[6] Center for Applied Internet Data Analysis - CAIDA. http://www.caida.org.

[7] Dalvik Virtual Machine. http://www.dalvikvm.com/, retrieved: Sept., 2017.

[8] eBay. http://www.ebay.com/, retrieved: Sept., 2017.

[9] Eicar test virus. http://www.eicar.org/86-0-Intended-use.html, retrieved: Sept., 2017.

[10] Erik Hjelmvik: The SPID Algorithm -Statistical Protocol IDentification. https://sourceforge.net/projects/spid/, retrieved: Sept., 2017.

[11] Expect. http://sourceforge.net/projects/expect/, retrieved: Sept., 2017.

[12] Floodlight. http://www.projectfloodlight.org/, retrieved: Sept., 2017.

[13] IANA.TCP and UDP port numbers, http://www.iana.org/assignments/port-numbers. retrieved: Sept., 2017.

[14] Intents and Intent Filters in Android. http://developer.android.com/guide/components/intents-filters.html, retrieved: Sept., 2017.

[15] Iperf. http://sourceforge.net/projects/iperf/, retrieved: Sept., 2017.

[16] Malware domain list. http://www.malwaredomainlist.com/, retrieved: Sept., 2017.

[17] MGEN. http://cs.itd.nrl.navy.mil/work/mgen/, retrieved: Sept., 2017.

[18] Microsoft Remote Desktop Connection. http://windows.microsoft.com/en-US/windows-vista/Connect-to-another-computer-using-Remote-Desktop-Connection, retrieved: Sept., 2017.

[19] MonkeyRunner. http://developer.android.com/tools/help/monkeyrunner-concepts.html, retrieved: Sept., 2017.

[20] MSN Messenger. http://explore.live.com/messenger, retrieved: Sept., 2017.

[21] nDPI. http://www.ntop.org/products/ndpi/, retrieved: Sept., 2017.

[22] NOX and POX SDN Controllers. http://www.noxrepo.org/, retrieved: Sept., 2017.

[23] Ookla Speedtest. http://www.speedtest.net/, retrieved: Sept., 2017.

[24] Open Network Operating System. http://www.onosproject.org, retrieved: Sept., 2017.

[25] OpenDayLight Project. http://www.opendaylight.org, retrieved: Sept., 2017.

[26] OpenDPI. https://github.com/thomasbhatia/OpenDPI, retrieved: Sept., 2017.

[27] Ostinato. http://ostinato.org/, retrieved: Sept., 2017.

[28] PsExec. http://technet.microsoft.com/en-us/sysinternals/bb897553, retrieved: Sept., 2017.

[29] Real VNC. http://www.realvnc.com, retrieved: Sept., 2017.

[30] RYU network operating system. http://osrg.github.com/ryu/, retrieved: Sept., 2017.

[31] tcpdump. http://www.tcpdump.org, retrieved: Sept., 2017.

[32] tcpreplay. http://tcpreplay.synfin.net/, retrieved: Sept., 2017.

[33] TG. http://www.postel.org/tg/tg.html, retrieved: Sept., 2017.

[34] TSTAT - TCP STatistic and Analysis Tool. http://tstat.polito.it/, retrieved: Sept., 2017.

[35] User Behavior based Emulator DEMO portal. http://ubetest1.hsnlab.tmit.bme.hu/, retrieved: Sept., 2017.

[36] uTorrent. http://www.utorrent.com/, retrieved: Sept., 2017.

[37] VPN Capture Android Application. https://play.google.com/store/apps/details?id=hu.edudroid.measurement_uploader, retrieved: Sept., 2017.

[38] Watir. http://www.watir.com/, retrieved: Sept., 2017.

[39] World of Warcraft. http://www.worldofwarcraft.com/index.xml, retrieved: Sept., 2017.

[40] Giuseppe Aceto, Alessio Botta, Antonio Pescapé, and Maurizio D'Arienzo. Unified architecture for network measurement: The case of available bandwidth. *J. Network and Computer Applications*, 35(5):1402–1414, 2012.

[41] Kanak Agarwal, Eric Rozner, Colin Dixon, and John Carter. Sdn traceroute: Tracing sdn forwarding without changing network behavior. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, pages 145–150, 2014.

[42] Ian F Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30, 2014.

[43] Ahmed Al-Jawad, Ramona Trestian, Purav Shah, and Orhan Gemikonakli. Baprobsdn: A probabilistic-based qos routing mechanism for software defined networks. In *Proc. 1st IEEE Conference on Network Softwarization (NetSoft), 2015*, pages 1–5. IEEE, 2015.

[44] Shane Alcock and Richard Nelson. Application flow control in youtube video streams. *SIGCOMM Comput. Commun. Rev.*, 41(2):24–30, April 2011.

[45] Shane Alcock and Richard Nelson. Measuring the accuracy of open-source payload-based traffic classifiers using popular internet applications. In *2013 IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops)*, pages 956–963, Oct 2013.

[46] G. Antichi, A. Di Pietro, D. Ficara, S. Giordano, G. Procissi, and F. Vitucci. Bruno: A high performance traffic generator for network processor. In *Proc. SPECTS '08*, pages 526–533, Edinburgh, UK, June 2008.

[47] G. Antichi, A. Di Pietro, D. Ficara, S. Giordano, G. Procissi, and F. Vitucci. Bruno: A high performance traffic generator for network processor. In *Performance Evaluation of Computer and Telecommunication Systems, 2008. SPECTS 2008. International Symposium on*, pages 526 –533, June 2008.

[48] S. A. Baset and H. Schulzrinne. An Analyis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proc. IEEE INFOCOM*, Barcelona, SPAIN, April 2006.

[49] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic Classification On The Fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, 2006.

[50] A. Botta, D. Emma, A. Pescape, and Giorgio Ventre. Systematic performance modeling and characterization of heterogeneous ip networks. In *Proc. of the 11th International Conference on Parallel and Distributed Systems*, volume 2, pages 120–124, July 2005.

[51] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. Do you trust your software-based traffic generator? *IEEE Communications Magazine*, 48(9), 2010.

[52] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 56(15):3531 – 3547, 2012.

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

[54] Zdravko Bozakov, Amr Rizk, Divyashri Bhat, and Michael Zink. Measurement-based flow characterization in centrally controlled networks. In *Proc. IEEE INFOCOM 2016*, pages 1–9. IEEE, 2016.

[55] Nevil Brownlee and Kimberly C Claffy. Understanding internet traffic streams: Dragonflies and tortoises. *IEEE Communications magazine*, 40(10):110–117, 2002.

[56] Iris Bueno, José Ignacio Aznar, Eduard Escalona, Jordi Ferrer, and Joan Antoni Garcia-Espin. An opennaas based sdn framework for dynamic qos control. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7. IEEE, 2013.

[57] Arthur Callado, Carlos Kamienski, Géza Szabo, Balazs Peter Gero, Judith Kelner, Stenio Fernandes, and Djamel Sadok. A survey on internet traffic identification. *Communications Surveys Tutorials, IEEE*, 11(3):37–52, 2009.

[58] Valentín Carela-Español, Pere Barlet-Ros, Albert Bifet, and Kensuke Fukuda. A streaming flow-based technique for traffic classification applied to 12 + 1 years of internet traffic. *Telecommunication Systems*, 63(2):191–204, 2016.

[59] Cihat Cetinkaya, Yalcin Ozveren, and Muge Sayit. An sdn-assisted system design for improving performance of svc-dash. In *Proc. Federated Conference on Computer Science and Information Systems (FedCSIS), 2015*, pages 819–826. IEEE, 2015.

[60] Wu chang Feng, F. Chang, Wu chi Feng, and J. Walpole. A traffic characterization of popular on-line games. *IEEE/ACM Transactions on Networking*, 13(3):488–500, June 2005.

[61] Xu Chen, J. Andersen, Z.M. Mao, M. Bailey, and J. Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC 2008.*, pages 177 –186, June 2008.

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

[63] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review*, 41(4):254–265, 2011.

[64] Alberto Dainotti, Alessio Botta, Antonio Pescapé, and Giorgio Ventre. Searching for invariants in network games traffic. In *Proc. of the 2006 ACM CoNEXT Conference*, CoNEXT '06, 2006.

[65] Alberto Dainotti, Antonio Pescape, and K.C. Claffy. Issues and future directions in traffic classification. *Network, IEEE*, 26(1):35—40, January 2012.

[66] Donato Emma, Antonio Pescape, and Giorgio Ventre. Analysis and experimentation of an open distributed platform for synthetic traffic generation. In *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, pages 277–283. IEEE, 2004.

[67] Ericsson. Mobility Report, Nov. 2016.

[68] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003.

[69] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *Proc. of the 10th ACM SIGCOMM conference on Internet measurement*, pages 281–287, New York, NY, USA, 2010. ACM.

[70] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: An intellectual history of programmable networks. *SIGCOMM Computer Communication Review*, 44(2):87–98, April 2014.

[71] W. Feng, A. Goel, A. Bezzaz, W. Feng, and J. Walpole. Tcpivo: A high-performance packet replay engine. In *Proc. of the ACM SIGCOMM workshop on Models, methods*, pages 57–64, 2003.

[72] Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACM Transaction on Networking*, 9(4):392–403, Aug. 2001.

[73] F. Gringoli, Luca Salgarelli, M. Dusi, N. Cascarano, F. Risso, and K. C. Claffy. Gt: Picking up the truth from the ground for internet traffic. *SIGCOMM Comput. Commun. Rev.*, 39(5):12–18, October 2009.

[74] Cesar D. Guerrero and Miguel A. Labrador. Traceband: A fast, low overhead and accurate tool for available bandwidth estimation and monitoring. *Computer Networks*, 54(6):977–990, 2010.

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

[76] Malati Hegde, MK Narana, and Anurag Kumar. Netmon: An snmp based network performance monitoring tool for packet data networks. *IETE Journal of Research*, 46(1-2):15–25, 2000.

[77] Harri Holma and Antti Toskala. *HSDPA/HSUPA for UMTS*. John Wiley & Sons, Ltd, 2006.

[78] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. *IEEE/ACM Transaction on Networking*, 11(4):537–549, Aug. 2003.

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

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

[81] Kun-chan Lan and John Heidemann. A measurement study of correlations of internet flow characteristics. *Computer Networks*, 50(1):46–62, 2006.

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

[83] J. Lindblom. A sinusoidal voice over packet coder tailored for the frame-erasure channel. *IEEE Transactions on Speech and Audio Processing*, 13(5):787 – 798, Sept 2005.

[84] Peng Liu, Fang Liu, and Zhenming Lei. Model of network traffic based on network applications and network users. In *Computer Science and Computational Technology, 2008. ISCSCT'08. International Symposium on*, volume 2, pages 171–174. IEEE, 2008.

[85] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. In *in Proc. of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 90–102, 2009.

[86] Gregor Maier, Fabian Schneider, and Anja Feldmann. A first look at mobile hand-held device traffic. In *Passive and Active Measurement*, volume 6032 of *Lecture Notes in Computer Science*, pages 161–170. Springer Berlin / Heidelberg, 2010.

[87] Natalia M Markovich and Jorma Kilpi. Bivariate statistical analysis of tcp-flow sizes and durations. *Annals of Operations Research*, 170(1):199–216, 2009.

[88] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communnication Review*, 38(2):69–74, March 2008.

[89] Péter Megyesi and Sándor Molnár. Finding typical internet user behaviors. In *Proc. 18th EUNICE Conference on Information and Communications Technologies*, pages 321–327, Aug 2012.

[90] Tal Mizrahi and Yoram Moses. Using reverseptp to distribute time in software defined networks. In *Proc. 2014 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 112–117. IEEE, 2014.

[91] Sándor Molnár and Zoltán Móczár. Three-dimensional characterization of internet flows. In *IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2011.

[92] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

[93] P. Abry, P. Flandrin, M. S. Taqqu, and D. Veitch. *Wavelets for the analysis, estimation, and synthesis of scaling data.* Wiley, 2000.

[94] Konstantina Papagiannaki, Nina Taft, Supratik Bhattacharyya, Patrick Thiran, Kavé Salamatian, and Christophe Diot. A pragmatic definition of elephants in internet backbone traffic. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 175–176. ACM, 2002.

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

[96] Kevin Phemius and Mathieu Bouet. "monitoring latency with openflow". In *9th International Conference on Network and Service Management (CNSM)*, pages 122–125, 2013.

[97] Marcin Pietrzyk, Louis Plissonneau, Guillaume Urvoy-Keller, and Taoufik En-Najjary. On profiling residential customers. In *International Workshop on Traffic Monitoring and Analysis*, pages 1–14. Springer, 2011.

[98] Ravi Prasad, Constantinos Dovrolis, Margaret Murray, and KC Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE network*, 17(6):27–35, 2003.

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

[100] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. Pathchirp: Efficient available bandwidth estimation for network paths. In *Proc. Passive and active measurements workshop*, 2003.

[101] Shriram Sarvotham, Rudolf Riedi, and Richard Baraniuk. Connection-level analysis and modeling of network traffic. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 99–103. ACM, 2001.

[102] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pages 68–81, 2004.

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

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

[105] Raúl Suárez, David Rincón, and Sebastià Sallent. Extending openflow for sdn-enabled synchronous ethernet networks. In *Proc. 1st IEEE Conference on Network Softwarization (NetSoft), 2015*, pages 1–6. IEEE, 2015.

[106] Srikanth Sundaresan, Walter De Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. Broadband internet performance: a view from the gateway. In *ACM SIGCOMM computer communication review*, volume 41, pages 134–145, 2011.

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

[108] G. Szabó, D. Orincsay, I. Szabó, and Sz. Malomsoky. On the validation of traffic classification algorithms. In *Proc. PAM*, Cleveland, Ohio, USA, April 2008.

[109] G. Szabó, Z. Turányi, L. Toka, S. Molnár, and A. Santos. Automatic Protocol Signature Generation Framework for Deep Packet Inspection. In *Proc. Valuetools*, Cachan, France, May 2011.

[110] Kevin Thompson, Gregory J Miller, and Rick Wilder. Wide-area internet traffic patterns and characteristics. *IEEE network*, 11(6):10–23, 1997.

[111] Tomasz Bujlow and Valentin Carela-Espanol and Pere Barlet-Ros. Independent comparison of popular DPI tools for traffic classification. *Computer Networks*, 76:75–89, 2015.

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

[113] K.V. Vishwanath and A. Vahdat. Swing: Realistic and responsive network traffic generation. *Networking, IEEE/ACM Transactions on*, 17(3):712 –725, June 2009.

[114] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay Kevin, and F. D. Smith. Tmix: A tool For Generating Realistic TCP Application Workloads In ns-2. *SIGCOMM Comput. Commun. Rev.*, 36(3):65–76, July 2006.

[115] Charles V. Wright, Christopher Connelly, Timothy Braje, Jesse C. Rabek, Lee M. Rossey, and Robert K. Cunningham. Generating Client Workloads and High-fidelity Network Traffic for Controllable, Repeatable Experiments in Computer Security. In *Proc. of the 13th international conference on Recent advances in intrusion detection, RAID'10*, pages 218–237, Ottawa, Canada, Sept 2010.

[116] T. Ye, D. Veitch, G. Iannaccone, and S. Bhattacharya. Divide and conquer: Pc-based packet trace replay at oc-48 speeds. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pages 262 – 271, feb. 2005.

[117] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and HarshaV. Madhyastha. Flowsense: Monitoring network utilization with zero measurement cost. In *Passive and Active Measurement*, volume 7799 of *Lecture Notes in Computer Science*, pages 31–41, 2013.

[118] Marcia Zangrilli and Bruce B Lowekamp. Using passive traces of application traffic in a network monitoring system. In *Proc. 13th IEEE International Symposium on High performance Distributed Computing, 2004*, pages 77–86. IEEE, 2004.

Publications

Journal papers (11.3 p)

[J1] **P. Megyesi**, G. Szabó, S. Molnár. User Behavior Based Traffic Emulator: A Framework for Generating Test Data for DPI Tools. *Computer Networks*, Vol 92, pp. 41-54, 2015. **(6/2=3 p)**

[J2] **P. Megyesi**, A. Botta, G. Aceto, A. Pescape, S. Molnár. Challenges and solution for measuring available bandwidth in software defined networks. *Computer Communications*, Vol 99, pp. 48-61, 2017. **(6/4=1.5 p)**

[J3] **P. Megyesi**, S. Molnár. Action Descriptive Strings. *Acta Universitatis Sapientiae-Electrical and Mechanical Engineering*, Vol 3:5–14, 2011. **(6/1=6 p)**

[J4] D. Szabó, A. Csoma, **P. Megyesi**, A. Gulyás, F. H.P. Fitzek. Network Coding as a Software Defined Networking Service. *Infocommunications Journal*, Issue 4, 2015. **(4/5=0.8 p)**

Conference papers (13.5 p)

[C1] S. Molnár, **P. Megyesi**, G. Szabó. Multi-functional Traffic Generation Framework Based on Accurate User Behavior Emulation In *Proc. IEEE INFOCOM (Demo)*, Turin, Italy, April 2013. **(0 p)**

[C2] S. Molnár, **P. Megyesi**, G. Szabó. Multi-Functional Emulator for Traffic Analysis In *Proc. IEEE ICC*, Budapest, Hungary, June 2013. **(3/2=1.5 p)**

[C3] **P. Megyesi**, S. Molnár Finding Typical Internet User Behaviors. In *Proc. 18th EUNICE Conference on Information and Communications Technologies*, Budapest, Hungary, Aug. 2012. **(3/1=3 p)**

[C4] S. Molnár, **P. Megyesi**, G. Szabó. How to Validate Traffic Generators? In *Proc. The 1st IEEE Workshop on Traffic Identification and Classification for Advanced Network Services and Scenarios (TRICANS)*, Budapest, Hungary, June 2013. **(3/2=1.5 p)**

[C5] **P. Megyesi**, S. Molnár. Forgalomemuláló Keretrendszer Tervezése és Fejlesztése In *Mester-próba Tudományos Konferencia (in Hungarian)*, Budapest, Hungary, May 2012. **(1/1=1 p)**

[C6] **P. Megyesi**, S. Molnár. Analysis of Elephant Users in Broadband Network Traffic In *Proc. 19th EUNICE Workshop on Advances in Communication Networking*, Chemnitz, Germany, 28-30 Aug. 2013. **(3/1=3 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)**

[C8] **P. Megyesi**, A. Botta, G. Aceto, A. Pescape, S. Molnár. Available Bandwidth Measurement in Software Defined Networks. In *Proc. ACM SAC 2016, NET - Networking Track, Pisa (Italy)*, 4-8 April, 2016. **(3/4=0.75 p)**

[C9] **P. Megyesi**, Zs. Krämer, S. Molnár. How Quick is QUIC? In *IEEE ICC*, Kuala Lumpur, Malaysia, 23-27 May 2016. **(3/2=1.5 p)**

[C11] Zs. Krämer, **P. Megyesi**, S. Molnár. A Google új, kísérleti QUIC protokolljának teljesítményelemzése. In *HTE Medianet 2015 (in Hungarian)*, Kecskemét, Hungary, Oct. 2015. **(1/2=0.5 p)**

[C11] S. Molnár, **P. Megyesi**, Sz. Solymos, Zs. Krämer, Z. Móczár. Flexible media transport framework for android. In *IEEE International Conference on Multimedia and Expo (ICME Demo)*, Seattle, WA, USA, 11-15 July 2016. **(0 p)**

Independent Citations

[1-J1] N. Sharma, D. Souryendu. Social fairness and channel loading effects in peer-to-peer connected networks. Peer-to-Peer Networking and Applications, 2015.

[2-J1] Stenio Fernandes. Principles of Performance Evaluation of Computer Networks. . Performance Evaluation for Network Services, Systems and Protocols, pp. 1–43. Springer International Publishing, 2017.

[3-J1] Stenio Fernandes. Internet Traffic Profiling. Performance Evaluation for Network Services, Systems and Protocols, pp. 113–152. Springer International Publishing, 2017.

[4-J4] D. Vukobratovic et al. CONDENSE: A reconfigurable knowledge acquisition architecture for future 5G IoT. IEEE Access Vol. 4, 2016.

[5-J4] D. Vukobratovic et al. Network function computation as a service in future 5G machine type communications. IEEE International Symposium on Turbo Codes and Iterative Information Processing, 4, 2016.

[6-C2] H. Cui et al. Streaming media traffic characterizations analysis in mobile Internet. International Symposium on Wireless Personal Multimedia Communications (WPMC), 2014.

[7-C2] M. Kacic et al. Traffic generator based on behavioral pattern. 9th International Conference for Internet Technology and Secured Transactions (ICITST), 2014.

[8-C3] T. Bujlow, V. Carela-Español, P. Barlet-Ros. Independent Comparison of Popular DPI Tools for Traffic Classification. Computer Networks, vol. 76, January, 2015.

[9-C4] S. Srivastava et al. Comparative study of various traffic generator tools. Recent Advances in. IEEE Engineering and Computational Sciences (RAECS), 2014.

[10-C4] Y. Han et al. Flow-level traffic matrix generation for various data center networks. Network Operations and Management Symposium (NOMS), 2014.

[11-C4] Y. Han et al. A Traffic Generation Method for Data Center Network Traffic Using Poisson Shot-Noise Model (in Korean). KNOM Review 17, no. 1.

[12-C4] Y. Han et al. Poisson Shot-Noise Process Based Flow-Level Traffic Matrix Generation for Data Center Networks. IFIP/IEEE International Symposium on Integrated Network Management, 2015.

[13-C4] B. Bylund, B. Nicklas Design and Implementation of a Traffic Generator using Unified Traffic Modelling. Final Thesis, Linkopings Universitet, 2015.

[14-C4] S. Srivastava et al. Evaluation of traffic generators over a 40Gbps link. Asia-Pacific Conference on Computer Aided System Engineering (APCASE), 2014.

[15-C4] P. Sawant et al. Traffic Generator for User Defined Payload and to Replay Pcap Files. Automation and Autonomous System 7.6, 2015.

[16-C4] M. Mosavi et al. Optimal Choice of Random Variables in D-ITG Traffic Generating Tool using Evolutionary Algorithms. Iranian Journal of Electrical & Electronic Engineering 11.2, 2015.

[17-C4] J. Cao et al. The investigation and implementation of packet generator with variable traffic and packet size on net FPGA. IEEE International Conference on Communication Technology, 2015.

[18-C4] A.G. Patil et al. Survey of synthetic traffic generators. IEEE International Conference on Inventive Computation Technologies, 2016.

[19-C4] A.G. Patil et al. Classification of UTGen synthetic traffic generator. IEEE Conference on Advances in Signal Processing, 2016.

[20-C4] T. Weiher. Vergleich von Hardware-und Software-Traffic-Generatoren und ihrem Einsatz in der Praxis. Seminars in Future Internet, Innovative Internet Technologies and Mobile Communications, 2016.

[21-C4] J. Nilsson. Cost-effective Packet Generation for Performance Evaluation of Network Equipment. MSc Thesis, Umea University, 2016.

[22-C6] T. Jin et al. Characterization of high-rate large-sized flows. White Paper, University of Virginia, 2014.

[23-C6] L. Polčák. Challenges in Identification in Future Computer Networks. ICETE 2014 Doctoral Consortium, 2014.

[24-C6] R. Addanki et al. A measurement-based study of big-data movement. European Conference on Networks and Communications (EuCNC), 2015.

[25-C6] Z. Feng et al. BLOC: A generic resource allocation framework for hybrid packet/circuit-switched networks. Journal of Optical Communications and Networking 8:9, 2016.

[26-C7] R. Kaur et al. Investigation of Performance Analysis of EDFA Amplifier Using Different Pump Wavelengths and Powers. International Journal of Advanced Research in Electronics and Communication Engineering 5:8, 2016.

[27-C7] Tomas Horvath, Radko Krkos, Lubos Dubravec. Deep data analysis in gigabit passive optical networks. Optica Applicata 47.1, pp. 157–170, 2017.

[28-C8] J.M. Jimenez et al. Study of Multimedia Delivery over Software Defined Networks. Network Protocols and Algorithms 7:4, 2016.

[29-C8] Y.R.M. Benisha. An Extensive Review on Software Defined Networking (SDN) Technologies. Journal of Advances in Electronics and Communication Engineering 2:1, 2016.