Measurement-Based Link Dimensioning for the Future Internet

Ricardo de Oliveira Schmidt

Measurement-Based Link Dimensioning for the Future Internet

Ricardo de Oliveira Schmidt

Graduation committee:

Chairman:	Prof. Dr.ir. Job van Amerongen
Promoter:	Prof. Dr.ir. Aiko Pras
Co-promoter:	Prof. Dr. Hans van den Berg

Members:

Prof. Dr. Lisandro Z. Granville	Federal University of Rio Grande do Sul, Brazil
Prof. Dr. George Pavlou	University College London, UK
Dr. Ramin Sadre	Université Catholique de Louvain, Belgium
Prof. Dr.ir. Lambert Nieuwenhuis	University of Twente, The Netherlands
Prof. Dr.ir. Boudewijn Haverkort	University of Twente, The Netherlands
Dr. Anna Sperotto	University of Twente, The Netherlands

Funding sources:
EU FP7 UniverSelf – 257513
EU FP7 Flamingo Network of Excellence – 318488
EU FP7 Mobile Cloud Networking – 318109
SURFnet's GigaPort3 project for Next-Generation Networks

CTIT Ph.D. thesis Series No. 14-334
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE
Enschede, the Netherlands

ISBN 978-90-365-3798-8
ISSN 1381-3617 (CTIT Ph.D. thesis Series No. 14-334)
DOI 10.3990/1.9789036537988
http://dx.doi.org/10.3990/1.9789036537988

Type set with LaTeX. Printed by Gildeprint Drukkerijen.

MEASUREMENT-BASED LINK DIMENSIONING FOR THE FUTURE INTERNET

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties,
in het openbaar te verdedigen
op woensdag 26 november 2014 om 16:45 uur

door

Ricardo de Oliveira Schmidt

geboren op 02 april 1985
te Passo Fundo-RS, Brazilië

Dit proefschrift is goedgekeurd door:
Prof. Dr. ir. Aiko Pras (promotor)
Prof. Dr. Hans van den Berg (co-promotor)

Look again at that dot. That's here. That's home. That's us. On it everyone you love, everyone you know, everyone you ever heard of, every human being who ever was, lived out their lives. The aggregate of our joy and suffering, thousands of confident religions, ideologies, and economic doctrines, every hunter and forager, every hero and coward, every creator and destroyer of civilization, every king and peasant, every young couple in love, every mother and father, hopeful child, inventor and explorer, every teacher of morals, every corrupt politician, every 'superstar,' every 'supreme leader,' every saint and sinner in the history of our species lived there – on a mote of dust suspended in a sunbeam.

— Carl Sagan,
Pale Blue Dot: A Vision of the Human Future in Space, 1994

Acknowledgments

My sincere thanks to all that somehow were involved in this PhD thesis or that were part of my life during the last four years. Special thanks to my supervisors Aiko and Hans for guiding me during the PhD research, and to my family for supporting my professional decisions.

And of course, I could not forget the little green man from Mars: thank you.

Abstract

Network operators have observed a significant increase in traffic demand in the past decade. That is because the Internet is now ubiquitous and provides means to access services essential to our daily life. To accommodate these traffic demands, operators over-provision their networks using simple rules of thumb for link dimensioning. However, throwing more link capacity in the network is not always a viable solution due to operational and financial constraints. Although the amount of link resources will likely not be a problem in the future Internet, the management of these resources will become more important. The current trend on virtualizing services and networks enables us to foresee how virtualization will soon dominate the Internet. Network operators will still own most of the physical infrastructure, but end users will be directly connected to companies that control essential online services and retain users' content. These companies are often referred to as the Internet *big players*. Virtual networks will enable transparent and seamless connection between end users and big players. The coexistence of many virtual networks on top of a single physical infrastructure will push for more sophisticated approaches to fairly share and allocate network resources. Efficient and accurate link dimensioning approaches can certainly make the difference in this context. Such approaches can (i) support operators on the optimal allocation of their link resources, while (ii) ensuring that Quality of Service metrics agreed with the big players are met, ultimately (iii) providing end users with good levels of Quality of Experience.

Focusing on proper allocation of link resources in the future Internet, in this thesis we develop and validate approaches for link dimensioning that are easy-to-use and accurate. Our starting point is an accurate and already validated dimensioning formula from previous works, which requires traffic statistics that can be calculated from continuous packet captures. However, packet captures are expensive and often demand dedicated hardware/software. Our approaches are able to estimate needed traffic statistics from coarser measurement data, namely sampled packets and flow-level measurements. Technologies able to provide us with such measurement data are largely available in network devices nowadays, namely sFlow, NetFlow/IPFIX and the more recent OpenFlow. The main contributions of this thesis can be divided in three parts.

The dimensioning formula we use is built upon the assumption of Gaussian traffic. In the past few years the advent of new online services, from social networking to online storage and video streaming, reshaped the behavior of network users. Past works that assessed Gaussian character of traffic relied on data measured relatively long ago, before these new services became highly popular. Therefore, our first contribution is an extensive investigation of the Gaussian character of current network traffic. We show that the assumption of Gaussian traffic remains valid and, hence, the dimensioning formula is still applicable to today's traffic. Moreover, in contrast to conclusions from previous works, we proved that traffic Gaussianity is closely related to measured traffic rates and independent of the number of simultaneously active hosts.

Aiming at ease of use, our proposed approaches for link dimensioning use data measured with largely available technologies in today's network devices. These technologies provide coarser data than plain packet captures, but also give us much more information than, *e.g.,* interface counters. As the second contribution of this thesis, therefore, we develop and validate approaches to estimate traffic statistics needed for the dimensioning formula from coarser traffic measurement data. In particular, we develop approaches to estimate traffic statistics from sampled packets obtained from sFlow, or similar packet sampling tools. These approaches account for the missing information (*i.e.,* skipped packets) and the random nature of the sampling algorithms. We also propose approaches that overcome the problem of data aggregation in flow-level measurements from NetFlow/IPFIX, or similar tools. To estimate the needed traffic statistics from flows, these flow-based approaches account for the missing information on individual packets. The proposed approaches in this thesis are able to accurately estimate required capacity at timescales from milliseconds to seconds.

Finally, the recent Software-Defined Networking (SDN) architecture claims to be ideal for managing dynamic network applications. OpenFlow is the best known enabler of SDN and it is already widely available in network devices. Although OpenFlow is primarily a traffic forwarding technology, in theory, it can also measure flow data as needed by our flow-based link dimensioning approaches (*i.e.,* NetFlow/IPFIX style). In practice, however, measured data from current implementations of OpenFlow are of poor quality. As the third contribution of this thesis, we introduce an approach to retrieve measured data from the OpenFlow switch, using the OpenFlow protocol, for purposes of link dimensioning. In addition, we assess the quality of measured data from OpenFlow both in a physical setup, using a real OpenFlow switch, and in a virtual setup, running a commonly used open source OpenFlow implementation. Results collected from our experiments lead us to conclude that measured data in OpenFlow is not yet suitable for link dimensioning.

Contents

For millions of years mankind lived just like the animals. Then something happened which unleashed the power of our imagination. We learned to talk.

— Stephen Hawking, 1994
In: Keep Talking, Pink Floyd.

CHAPTER 1

Introduction

This chapter presents background information and motivates the research of this Ph.D. thesis, details our research goal and questions, and outlines the thesis structure.

1.1 Background

The Internet has become an essential tool for the modern society. It is ubiquitous and offers a plethora of online services accessible via a huge diversity of interconnected devices. Network operators need to cope with the "ever increasing" demand of network traffic. Figure 1.1 gives an idea of today's traffic volume and enables us to image what traffic demands for the near future will be. This figure shows the total volume of in/out transit traffic at the Amsterdam Internet Exchange (AMS-IX) in the past years, which clearly has experienced an exponential-like growth from 2.7 PB in 2002 to 1.2 EB (exabytes) in 2014[1].

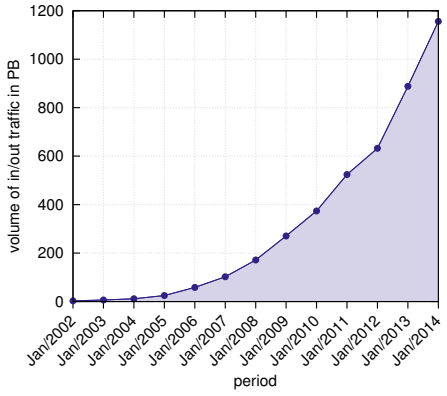


Figure 1.1: Volume of in/out traffic at AMS-IX from 2002 to 2014.

[1] https://www.ams-ix.net/technical/statistics

In addition to the increasing traffic demands, the somehow disorganized (non-structured) growth of the Internet comes to add complexity to the network management. In some cases, traffic demands can be provisioned by simply throwing in more resources into the network infrastructure. However, this might not always be a viable solution due to, *e.g.,* financial or operational constraints.

The trend towards virtualization of networks and services allows us to foresee that the complex task of managing networks in the future Internet will call for approaches to support optimal allocation of network resources. Next we describe what we envision as one of the scenarios of the future Internet, where excessive use of network virtualization will aim at seamless services for end users.

A scenario of the Future Internet

The Internet is becoming more and more dominated by a small number of *big players* [52]. These are companies that own essential online services, store users' content and are also dominating the mobile market (*e.g.,* Apple with iOS, Google with Android and Microsoft with Windows Phone). Retaining the content produced and shared by users, often compelling a fidelity relationship is what gives the big players the power to decide how the Internet should work. Examples of big players are Google, Microsoft, Akamai, Facebook, and the rising Dropbox and Netflix.

Our view of the future Internet is that the direct relationship between big players and the end users will narrow. Although network operators will still own (most of) the infrastructure, even the Internet access might become part of the services offered by big players. This will eliminate the intermediate relationship between end users and network operators.

Figure 1.2a shows how the relationship between the end user, network operator (in this example Deutsche Telekom) and big players (in this example Google and NetFlix) works nowadays. The only way the end user can reach services offered by the big players is by intermediately hiring the services of the network operator. In the future Internet, as shown in Figure 1.2b, end users will deal directly with big players and the access to services will be independent of an intermediate negotiation with a network operator. This scenario does resemble Virtual Private Networks (VPN) connections on top of physical infrastructures. The big players hire the access and transport infrastructure from network operators. This way the big players can create end-to-end connections between their customers and data centers. Ultimately, the big players end up creating their own ecosystems in the Internet by besieging their respective customers; and these Internet ecosystems can span over multiple operators domains. In fact, this scenario we envision is already taking shape at an initial scale. Companies, such as Google, are hiring huge amounts of infrastructure resources from

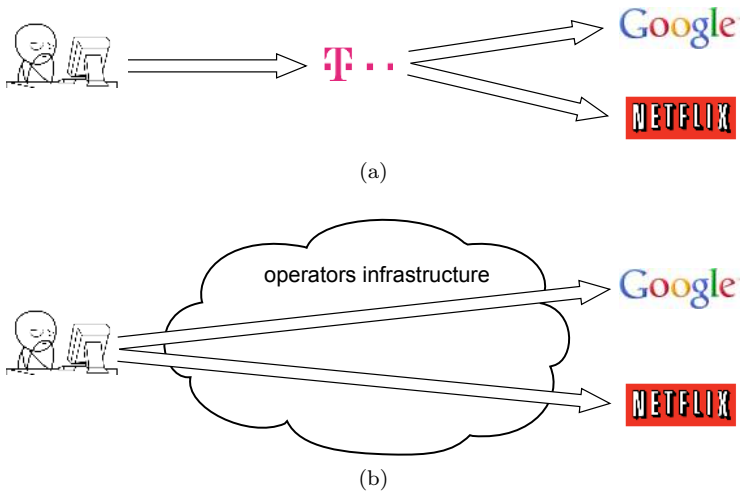


Figure 1.2: Relationship between end users, network operators and big players: (a) current scenario and (b) future Internet scenario.

operators, such as Deutsche Telekom, to provide connectivity between end users and data centers. Operators have been referring to these services as *Internet as a Service*.

Although infrastructure is available, there are many challenges still to be addressed so that the envisioned scenario can become reality on its full conception. These range from political to ethical, financial and technological challenges. Concerning technological challenges, network and services virtualization will be one of the underlying pillars enabling the future Internet. Resources of a single physical network will be shared among multiple coexisting virtual networks, which demands novel approaches for resource allocation. Link capacity is one of the main resources that must be fairly shared and allocated, and this can be achieved with sophisticated approaches for link dimensioning. Notice that all involved parts can benefit from efficient link dimensioning approaches: (i) these approaches can support proper use of operator's bandwidth resources; (ii) these approaches can help ensuring that traffic from/to a big player meets the Quality of Service (QoS) levels agreed with network operators; and (iii) as a consequence of a properly dimensioned network, end users can ultimately experience good levels of Quality of Experience (QoE).

In this thesis we address the link dimensioning problem. We propose approaches for link dimensioning that can accurately estimate required capacity of traffic by using traffic measurement technologies widely found at network op-

erators. In the next section we provide an overview of current approaches used for link provisioning and dimensioning, pointing our their pros and cons.

1.2 Link Dimensioning Overview

Link dimensioning is used by network operators to properly provision their network links according to the traffic demands. If traffic demands are higher than the allocated capacity, end users might experience network performance degradation due to packets loss caused by, *e.g.,* buffer overflow in network routers. Aiming at meeting desired QoS levels and, hence, avoiding violation of Service-Level Agreement (SLA), operators continuously monitor the bandwidth utilization of their links. Network operators commonly use well-stablished and widely deployed traffic monitoring and measurement tools. A typical approach combines the Simple Network Management Protocol (SNMP) [101] with the Multi Router Traffic Grapher (MRTG) [4] or Round-Robin Database Tool (RRD) [6]. The latter are used for storage and visualization purposes. SNMP allows for operators to access interface counters defined by Management Information Bases (MIB) and obtain information, such as the number of received and sent bytes by the interface since the device was last rebooted.

Cisco has published in [26] a how-to guideline for calculating bandwidth utilization using SNMP. The procedure is quite straightforward and relies on two counters defined by MIB-II [82], namely, `ifInOctets` and `ifOutOctets`[2]. These counters provide, respectively, the number of received and sent octets for a given network interface. According to Cisco's document, bandwidth utilization can be calculated by

$$\frac{\Delta \texttt{ifInOctets}}{T} + \frac{\Delta \texttt{ifOutOctets}}{T},$$

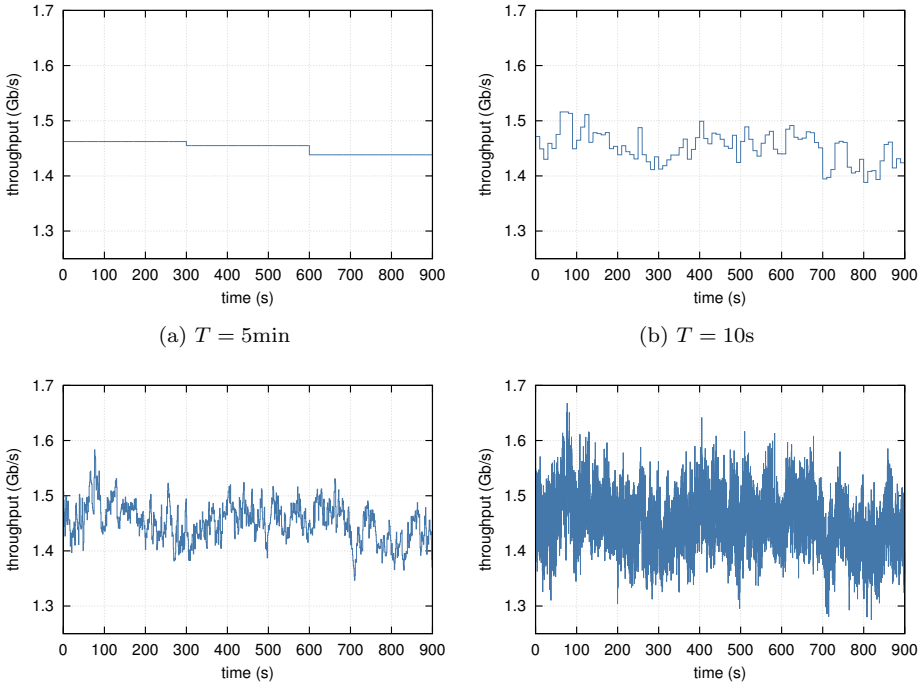
where T must be greater than zero and defines the size of the time interval between two consecutive readings of the octet counters, and Δ represents the modulus of the difference between the values of the counters read at times t_0 and $t_0 + T$.

Bandwidth utilization is typically calculated by polling interface counters every 5 to 15 minutes. Aiming at over-provisioning, the required link capacity is defined by adding a *safety margin* to the average bandwidth utilization. This safety margin might depend on several factors, such as period of the day and QoS requirements. Typically, operators define this safety margin as a simple percentage of the average bandwidth utilization [31]. This simplistic approach

[2]Since `ifInOctets` and `ifOutOctets` are 32-bit counters, which wraparound frequently, it would be better to use the equivalent 64-bit counters `ifHCInOctets` and `ifHCOutOctets`.

for defining the safety margin is often referred to as *rule of thumb* for over-provisioning. Given the wide availability of the SNMP protocol, rules of thumb approaches are easy-to-use.

One of the main problems with the rules of thumb approach is that traffic fluctuations are averaged within too large time bins. That is, the way to calculate bandwidth utilization might overlook traffic bursts that happen at smaller timescales, such as seconds or fraction of seconds. The overlooked bursts ultimately create problems for network performance and degrade user experience. This problem of averaging traffic in large time bins is shown in Figure 1.3. This figure shows the throughput time series of a 15-minute traffic trace generated using various values for T (bin size). It becomes clear how traffic fluctuations completely disappear when larger timescales are used. While the highest 5-minute peak observed in Figure 1.3a is 1.46 Gb/s, when setting $T = 10$s we can observe traffic rates up to 1.52 Gb/s. When measuring the traffic at the milliseconds timescale, $T = 100$ms, we observe rates reaching up to 1.68 Gb/s.



(a) $T = 5$min

(b) $T = 10$s

(c) $T = 1$s

(d) $T = 100$ms

Figure 1.3: Time series for a 15-minute long traffic capture at various timescales.

SNMP-based rules of thumb do not scale according to small timescales. For example, if the network operator is interested in provisioning the link at shorter timescales, it is not feasible to read SNMP counters every 100 ms. To account for that, operators tend to use rules of thumb with large safety margins that will likely overestimate the required link capacity at the timescale in question, hence, wasting link resources that could be allocated to other purposes. However, if even a large safety margin does not suffice, the link will be under-provisioned and, ultimately, performance degradations might be experienced by end users.

Many alternative approaches for link dimensioning have been proposed with the aim of being more intelligent and reliable than SNMP-based rules of thumb for over-provisioning. However, the higher accuracy of these approaches often comes at the cost of requiring more efforts on network traffic measurements. For example, the work in [109] defines a link dimensioning formula that requires traffic statistics (*e.g.,* traffic variance) usually calculated from packet-level measurements. That is, on having continuous packet capturing, one can have a complete overview of the transferred traffic and, therefore, calculate required link capacity with higher precision even at very short timescales. Theoretically, for such approaches, the limit on how small the timescale can be is actually dictated by the hardware/software that is used to capture packets.

Nonetheless, even providing estimations of required capacity with high accuracy, approaches that require continuous packet capturing are not attractive and typically not adopted by network operators. That is because traffic rates in high-speed links, and the ever increasing volume of traffic, make packet capturing operationally and financially unfeasible. Some works such as [70, 97] have addressed the challenge of packet capturing in high-speed links, *e.g.,* 10 to 100 Gb/s, by proposing the use of hardware acceleration techniques. These solutions demand very specific and mostly expensive hardware and software. Therefore, network operators stick to easy-to-use, though not reliable, SNMP-based rules of thumb.

In this thesis we aim at finding a tradeoff between ease of use and accuracy for link dimensioning. We make use of the accurate and already validated dimensioning formula proposed in [109]. However, instead of relying on costly packet captures, our approaches provide ways to compute the input parameters for the dimensioning formula from traffic measurement technologies that can be easily found at operators' networks, namely sFlow, NetFlow/IPFIX and OpenFlow.

Literature Review

This section provides a brief literature review on the problem of link dimensioning. Our decision to keep the literature review short is based on the fact

that very few novel steps were taken in this area since the work of [109]. For a more detailed literature review, therefore, one can refer to [109]. Also, in this section we focus mostly on measurement-oriented link dimensioning (opposed to model-based approaches), which is within the context of the research in this thesis.

Some of the proposed approaches for link dimensioning only address specific applications or metrics, and most of the applications require traffic measurements at the packet level, *i.e.,* continuous packet capturing. For example, the work in [96, 109, 112], which is further detailed in Section 1.4, proposes a dimensioning formula focusing on link rate exceedance that requires traffic statistics to be computed from packet measurements. In [80] the authors propose to estimate the same statistics from routers buffer occupancy. Although this second approach does not need on-link traffic measurements, it requires additional complexity to be implemented in the routers. The work in [112] proposes a provisioning procedure requiring minimal measurement effort, using minimal model assumptions, and with QoS constraints expressed in link rate exceedance. However, this work focuses on traffic variations that are solely due to fluctuations at the flow level, and the proposed bandwidth provisioning method is only valid for relatively large timescales, *e.g.,* 1 second.

In [74] the authors propose a bandwidth estimator based on a $M/G/\infty$ model. The main limitation of this work is, however, that it requires continuous packet-level measurements to observe packet arrivals and sizes. In addition, the model is further divided into four different sets of equations, and the selection on which one to use depends on the timescale the operator wishes to dimension a given link. This characteristic limits the flexibility given that the link dimensioning procedure needs to be adapted once the timescale is changed.

Other approaches use link dimensioning within more specific cases. For example, in [9] the authors proposed a bandwidth allocation procedure for delay sensitive applications along a path of point-to-point Multiprotocol Label Switching (MPLS) connections. Focusing on improving QoS, the approach in [49] accounts for packet delays for dimensioning links. But once again, the requirement of packet-level measurements comes to be the main drawback of these approaches.

Not only packet-based approaches have been proposed. Concerning flow-level traffic measurements, the authors in [10] propose a traffic model on Poisson flow arrivals and i.i.d. flow rates that is able to predict bandwidth consumption for non-congested backbone links, making assumptions on the evolution of traffic within single flows. The authors in [12] provide dimensioning formulas for IP access networks where QoS is measured by per-flow throughput. In such work, only elastic data traffic (*i.e.,* TCP connections) was considered.

As further detailed in the next section, in this thesis we propose link dimensioning approaches focusing on ease of use and accuracy. Our approaches do not put any constraint on the type of the traffic when calculating the required capacity of a given traffic aggregate.

1.3 Thesis Contribution

Figure 1.4 positions this thesis in relation to the currently used rules of thumb and the approach proposed in [109], from which we use the dimensioning formula as starting point of our research (the formula is further detailed in Section 1.4). This formula originally requires continuous packet capturing. In this thesis we investigate and develop alternative link dimensioning approaches with comparable accuracy as the work presented in [109], but also with comparable ease-of-use as the SNMP-based rules of thumb.

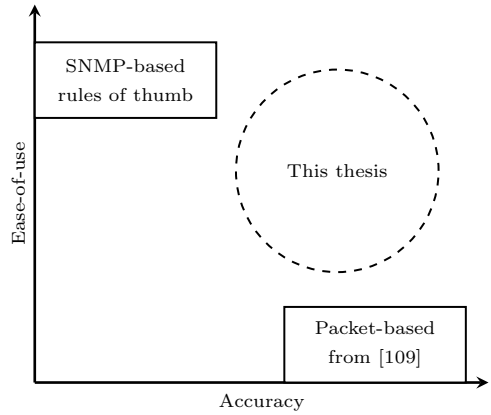


Figure 1.4: Position of this thesis.

From the easiness-of-use point of view, we avoid the use of continuous packet capturing and propose methods for calculating traffic statistics, required by the adopted dimensioning formula, from alternative and widely available traffic measurement technologies. The idea is to use measurement technologies that can easily be found at operators devices and, perhaps, are already used for other purposes than link dimensioning (*e.g.,* as presented in [105]). Measurement technologies we study in this thesis are sFlow and packet sampling, NetFlow/IPFIX flow-level measurements and the more recent OpenFlow. Although easier to use, these technologies provide coarser measurements than plain packet capturing and, consequently, accuracy of estimations of required capacity might be

imperiled. This problem is further addressed and discussed while validating the proposed methods in their respective chapters. Giving that this thesis takes the dimensioning formula from [109] as starting point, in the next section we present this formula in detail.

1.4 Link Dimensioning Formula

The link dimensioning formula used in this thesis was proposed in [109], and extensively validated in [80, 96, 112]. This formula aims at "link transparency", which means that end users should almost never perceive network performance degradations due to lack of bandwidth resources. To statistically assure link transparency to users, the provided link capacity C should satisfy

$$\mathbb{P}\{A(T) \geq CT\} \leq \varepsilon \,, \tag{1.1}$$

where $A(T)$ denotes the total amount of traffic arriving in intervals of length T, and ε indicates the probability that the traffic rate $A(T)/T$ is exceeding C at the timescale T.

The link dimensioning formula requires that traffic aggregates at timescale T are *Gaussian* (*i.e.,* $A(T)$ are normally distributed) and *stationary*. The link capacity $C(T, \varepsilon)$, needed to satisfy Eq. (1.1), can be calculated by

$$C(T, \varepsilon) = \rho + \frac{1}{T}\sqrt{-2\log(\varepsilon) \cdot \upsilon(T)} \,, \tag{1.2}$$

where the mean traffic rate ρ is added with a term that can be seen as a "safety margin". This term depends on the traffic variance $\upsilon(T)$ at the chosen timescale. Mean traffic rate ρ and traffic variance $\upsilon(T)$ are defined by, respectively

$$\rho = \frac{1}{nT}\sum_{i=1}^{n} A_i(T) \quad \text{and} \quad \upsilon(T) = \frac{1}{n-1}\sum_{n=1}^{n}(A_i(T) - \rho T)^2 \quad ,$$

where $A_i(T)$ is the amount (in bytes) of observed traffic in time interval i of length T and n the number of monitored intervals.

By including the traffic variance, the formula also accounts for traffic bursts that would potentially threaten link transparency requirements. Notice that this formula is very flexible: network operators can choose the timescale T and the exceedance probability ε according to the QoS that they want to provide to their customers. For example, while larger T (*i.e.,* around 1s) would be enough to provide good quality of experience to users on web browsing, shorter T (*i.e.,* milliseconds scale) should be chosen when real time applications, such as Voice over IP (VoIP), are predominant in the network, since the formula would be

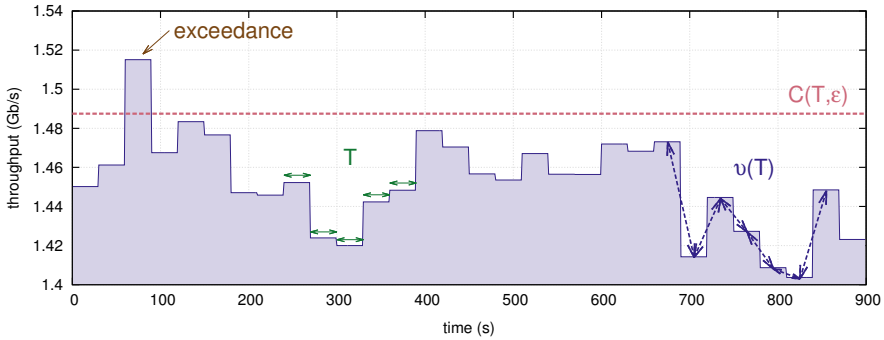


Figure 1.5: Visual explanation of the parameters of the link dimensioning formula of Equation (1.2); traffic time series created with $T = 30$s.

able to capture traffic bursts that happen at such short time scales. The value for ε should be chosen in accordance to the desired QoS. Roughly spoken, while T defines to which extent the duration of traffic fluctuations are important, ε accounts for how many intervals of size T the traffic aggregate $A(T)$ is allowed to be higher than the required bandwidth $C(T, \varepsilon)$. Notice that the choice of T is also related to the size of router buffers to accommodate traffic that exceeds link capacity.

To help the understanding of how the link dimensioning formula of Equation (1.2) works, Figure 1.5 provides a visual explanation of the formula's parameters. In this example, we use a 15-minute long traffic time series created with $T = 30$s. That is, the traffic is aggregated in time bins of size 30s, as illustrated in green arrows between 240s and 390s. The variance $v(T)$ comes from the difference between the traffic aggregates of each bin, as illustrated by the dark blue arrows between 660s and 870s. As mentioned before, the network operator must choose an appropriate value of ε according to the QoS to be provided. All these parameters, and the traffic average rate ρ, are applied to the Equation (1.2) and the $C(T, \varepsilon)$ is obtained. To determine whether the estimation is successful or not, one needs to inspect how many of the traffic bins have traffic rate that exceeds the estimated $C(T, \varepsilon)$. In the example of Figure 1.5 there is only one bin with rate higher than the estimated required capacity.

Figure 1.6 shows a practical example of the use of this dimensioning formula. The figure shows the time series of traffic throughput calculated from 15 minutes of continuous packet capturing. In this example, the size of time bins is set to $T = 1$s, *i.e.,* the time series shows the average traffic rate for every second during 15 minutes. In the dimensioning formula we set $\varepsilon = 1\%$. The example trace has $\rho = 1.45$ Gb/s and $v(T) = 1.21$ Gb. For a 15-minute trace, we have 900 time

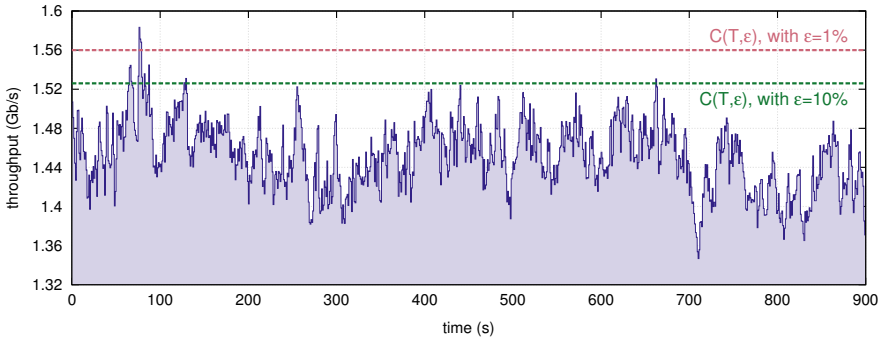


Figure 1.6: Estimated $C(T, \varepsilon)$ using the link dimensioning formula of Equation (1.2) for a sample 15-minute traffic trace; $T = 1$s and $\varepsilon = 1\%$ or $\varepsilon = 10\%$.

bins of size $T = 1$s. By setting $\varepsilon = 1\%$, we allow for up to 9 time bins to have a throughput higher than $C(T, \varepsilon)$. That is, the resulting estimation is successful if no more than 1% of time bins have throughput higher than $C(T, \varepsilon)$. In this example the formula estimated $C(T, \varepsilon) = 1.56$ Gb/s, which resulted in only 3 time bins with throughput exceeding this estimation. This also shows that the estimated $C(T, \varepsilon)$ is higher than the actual required capacity – which would yield an estimation such that exactly 9 time bins will have throughput higher than the estimated $C(T, \varepsilon)$. Nonetheless, we assume that in this example the overestimation is not excessively high. That is because excessive overestimation might result in cases that the throughput of no time bin is higher than the estimated $C(T, \varepsilon)$ for $\varepsilon > 0$.

For matters of illustration, in Figure 1.6 we also show the required capacity $C(T, \varepsilon)$ computed with $\varepsilon = 10\%$. In this case, by setting a larger exceedance probability ε, the estimated required capacity is lower than the previous one. Clearly, this happens because with a larger ε we accept that in more bins the traffic rate exceeds the estimated capacity. In this case, we allow for a total of 90 time bins to have rates higher than $C(T, \varepsilon)$, but actually only 13 bins exceed the estimated capacity of 1.53 Gb/s, which again gives us a successful estimation. Finally, for matters of comparison, simulating SNMP-based rules of thumb [31], by adding add 50% of the average throughput to itself ($\rho \cdot 1.5$), the estimated required capacity becomes 2.18 Gb/s, which overly overestimates the required capacity for this example trace.

If the network operator performs continuous packet capture, the calculated ρ and $v(T)$ will faithfully represent the real traffic statistics. However, these statistics may not be straightforwardly calculated from other measurement technologies, such as sampled packets or flow-level traffic measurements.

1.5 Goal, Research Questions & Approaches

1.5.1 Goal

As described in Section 1.1, link dimensioning is an important task performed by network operators to properly provision their network links. However, optimal allocation of resources in the excessively virtualized networks, as envisioned in the scenarios of future Internet, will call for more accurate approaches for link dimensioning than those currently used by network operators. Given that (i) network operators still stick to rough estimations obtained using old-fashioned rules of thumb, and (ii) more accurate approaches for link dimensioning often demand traffic measurements at the packet level, we define the overall goal of this thesis as the following:

> ***Research Goal:*** *Develop easy-to-use and accurate approaches to estimate required link capacity for purposes of link dimensioning.*

Unlike most related work on link dimensioning, this thesis does not propose new link dimensioning formulas. Instead, we adopt the dimensioning formula from [109], described in Section 1.4, and focus our efforts on investigating ways to calculate the parameters required by such formula (*i.e.,* mean traffic rate and traffic variance) from other types of measurements than continuous packet capturing. Addressing the property of being easy-to-use, we only consider measurement technologies largely found in network devices and that scale to high traffic rates observed nowadays. Next, we describe the research questions defined to achieve our overall goal.

1.5.2 Research Questions and their approaches

As mentioned in Section 1.4, the adopted link dimensioning formula from Equation (1.2) requires that traffic rates aggregated at a certain timescale follow a Gaussian process. Internet traffic has been evolving due to the recent advent of many online services, such as Facebook, Dropbox, Youtube and NetFlix. Such services transformed users behavior what, consequently, potentially reshaped network traffic. In the past, two main works have addressed the Gaussianity fit of traffic, for example [71] in 2002 and [110] in 2006. However, these works relied on data measured relatively long ago and it is important to assess the Gaussianity fit of traffic once again. Therefore, our first research question is defined as:

> ***RQ-1:*** *Given the importance of Gaussian characteristics for link dimensioning purposes, and the emergence of new online services, is current Internet traffic still Gaussian?*

We address the Research Question 1 in Chapter 2 by assessing whether the Gaussianity assumption of traffic still holds for current traffic. We do so by assessing the Gaussianity fit of an entire traffic dataset, comprising traffic measurement from around the globe. This dataset is later used in other chapters to validate our proposed link dimensioning procedures. In addition, we further study properties of (non-)Gaussian traffic aiming at finding what causes the lack of Gaussianity in certain traffic aggregates. This would allow operators to better judge whether their traffic is Gaussian or not, solely based on the mix of applications and hosts behavior, without the need for performing traffic measurements.

Concerning ease of use of the proposed link dimensioning approaches in this thesis, all the remaining research questions relate to investigating how to use widely available traffic measurement technologies for link dimensioning purposes. The main drawback of the proposed approach in [109] is that it requires continuous packet captures and this is not trivial to do on current high-speed links due to operational and financial constraints. The most straightforward solution to measure high amounts of traffic in an easier way is by reducing the measurement workload, which can be done by deploying packet sampling technologies. sFlow certainly is among the most deployed traffic monitoring and measurement technologies that implement packet sampling, and many network devices are sFlow-enabled[3]. For example, it is known that AMS-IX [65] and CERN [59] use sFlow to measure the traffic from their network, and such information can later be used to support network management operations. However, sampled packets only give us a partial overview of the actual observed traffic. Therefore, we must find ways to estimate traffic statistics that are needed by the link dimensioning formula (*i.e.,* average traffic rate and traffic variance) from sampled data. Given that, our second research question is defined as:

> ***RQ-2:*** *Given its potential for scalability at high-speed links, traffic measurement technologies that implement packet sampling are very attractive. The problem is that sampled data provides a partial view of the traffic transferred over the link. Therefore, how can we estimate traffic average rate and traffic variance, crucial inputs for the adopted dimensioning formula, from sampled packets?*

Research Question 2 is addressed in Chapter 3 where we investigate the impact of packet sampling on link dimensioning. In addition to the sampling algorithm implemented by sFlow, we study two other sampling algorithms, namely Bernoulli and n-in-N sampling. Although the estimation of the traffic average rate from sampled data is quite straightforward, to estimate the traffic variance

[3]http://www.sflow.org/products/index.php

might not be. We show that simply scaling up the variance calculated from sampled data might not yield the expected results, ultimately, impacting negatively on the results of the link dimensioning procedure. Therefore, we propose different formulas to estimate the traffic variance from sampled packets. Furthermore, we also show the impact of the exporting process of sampled packets, as implemented by sFlow, on the link dimensioning procedure.

Another very attractive traffic measurement technology is the flow-based one. This is mainly due to the wide deployment of, among others, the Cisco's NetFlow and IPFIX-based probes. Nowadays, many network devices are flow-enabled, which makes flows a commonly found measurement technology. Besides being largely available at operators networks, flows are a scalable measurement technology, providing aggregated view of measured traffic. However, its scalability advantage comes at the cost of lack of more granular information about the observed traffic. For example, from flows one can determine the duration and number of packets and bytes transferred between two hosts, but cannot infer the individual packet transmission times or sizes. Without the information on individual packets, the calculation of traffic variance becomes a real challenge. Therefore, our third research question is defined as:

> ***RQ-3:*** *Given the widespread availability of flow-enabled network devices, flow measurements are a very attractive technology with the additional advantage of being scalable for monitoring large amounts of traffic data. However, the summarized data provided by flows impose challenges on its use for link dimensioning purposes. Therefore, how can we estimate traffic average rate and variance without information on individual packets?*

The Research Question 3 is addressed in Chapter 4 and 5. A straightforward approach is described in Chapter 4, which builds traffic time series from flows and uses these time series to estimate traffic variance for later use in the dimensioning formula. This approach relies on the basic assumption that packets within flows are uniformly distributed and of the same size. Clearly, such assumptions hardly represent reality. Despite that, this simple approach is able to provide satisfactory results on estimating the required capacity at larger timescales. Relying solely on flows and based on optimistic assumptions about traffic, this approach is limited to estimate required link capacity at large timescales only. Therefore, in Chapter 5 we describe a procedure based on flow traffic models that is able to provide accurate estimations of required capacity at much smaller timescales, such as 1ms. The gain on accuracy comes at the cost of requiring parameters tuning, which makes this second flow-based procedure less easy-to-use than the first one.

We validate the proposed link dimensioning procedures from Chapters 3, 4 and 5 against an empirically defined ground-truth. That is possible because

our dataset consists solely of packet-level measurements, which enables us to empirically find the required capacity for the measured traffic given the values of T and ε of interest.

Finally, together with the advent of the Software-Defined Networking (SDN) paradigm, the protocol OpenFlow has recently gained lots of attention from both industry and academia, and it is getting adopted by network operators. OpenFlow is not primarily intended for traffic measurements, but it allows the decoupling of the control and data planes in a network. Similarly to NetFlow, OpenFlow is able to measure traffic on a per-flow basis, keeping counters with predefined information, such as number of packets and bytes. Therefore, OpenFlow can, in theory, provide traffic measurements for link dimensioning, which leads us to the fourth research question, defined as:

> ***RQ-4:*** *The increasing interest in SDN has made the recent OpenFlow largely implemented in many network devices. In theory, OpenFlow can measure traffic in a NetFlow/IPFIX style. Therefore, can we use OpenFlow per-flow traffic measurements for link dimensioning purposes?*

We address the Research Question 4 in Chapter 6. We introduce an approach to retrieve traffic measurements from the switch solely using messages defined by the OpenFlow protocol. These measurements consist of per-flow packet and byte counters maintained by the OpenFlow switch. OpenFlow is already largely available in network devices from different vendors. In Chapter 6 we assess the quality of the per-flow traffic measurements obtained from OpenFlow implementations in (i) a physical setup using a real OpenFlow switch, and (ii) a virtual setup using Open vSwitch, which serves as basis for many vendor OpenFlow implementations. We demonstrate that the tested implementations of OpenFlow do not provide traffic measurements of enough quality for link dimensioning. In fact, we show that the measurements lack accuracy even when measuring traffic aggregates from a single IP flow.

By answering the Research Questions 2, 3 and 4, we obtain several ways of estimating the required capacity for link dimensioning purposes, all using traffic measurement technologies that are widely found at operators' networks, combined with an extensively validated link dimensioning formula. Therefore, the results of this thesis provide operators with the opportunity to choose the most appropriate procedure so that their requirements are fulfilled.

1.6 Thesis Organization

Given the main goal, research questions and the approaches to answer these questions, in the following we provide a short summary of each chapter in the

remainder of this thesis. In addition, we link the publications used as the basis for each chapter. Figure 1.7 illustrates the position of chapters, serving also as a guideline throughout this thesis.

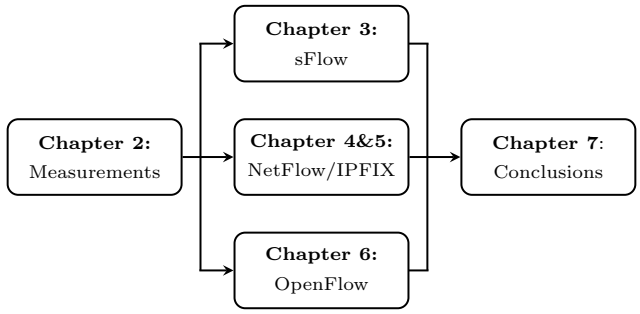


Figure 1.7: Thesis's organization.

Chapter 2: Datasets and Traffic Characteristics

In this chapter we introduce the traffic measurements dataset that comprises hundreds of packet-level traffic traces used throughout this thesis to validate our proposed link dimensioning approaches. Given the importance of Gaussianity fit of traffic in the link dimensioning approach of [109], which is the basis of our work, we present an extensive study on Gaussian characteristics of the traffic for the whole dataset. Among our findings in this chapter, we show the relationship between of Gaussianity fit and horizontal traffic aggregation (*i.e.,* defined by the size of the measurement interval). We demonstrate that it is safer to relate the degree of Gaussianity to the traffic average rate than to the number of active hosts. In addition, we also identify the relationship between network usage patterns and traffic Gaussianity fit. We verify the impact of abnormal traffic bursts on Gaussianity and further investigate applications and users behind these bursts. The Gaussianity study presented in Chapter 2 has been published on the following two publications:

- R. de O. Schmidt, R. Sadre and A. Pras, *Gaussian Traffic Revisited.* In Proceedings of the 12th IFIP Networking Conference, 2013 [41]

- R. de O. Schmidt, R. Sadre, N. Melnikov, J. Schönwälder and A. Pras, *Linking Network Usage Patterns to Traffic Gaussianity Fit.* In Proceedings of the 13th IFIP Networking Conference, 2014 [40]

Chapter 3: sFlow-based Link Dimensioning

Since the main drawback of the link dimensioning approach in [109] is the fact that it requires continuous packet captures, in this chapter we explore what we believe to be the first idea to come in mind in order to reduce the traffic measurement overhead for link dimensioning purposes: the deployment of packet sampling technologies. In particular we study three sampling algorithms: Bernoulli, n-in-N and the specific strategy implemented by sFlow. Besides being widely implemented within traffic measurement tools, Bernoulli and n-in-N are also described in [119]. We further study the impact of the exporting process implemented by the measurement tool sFlow on link dimensioning. We show that it is feasible to use packet sampling strategies to reduce measurement efforts and still have accurate estimations of required capacity. The content in this chapter is partially based on the following publication:

- R. de O. Schmidt, R. Sadre, A. Sperotto and A. Pras, *Lightweight Link Dimensioning using sFlow Sampling.* In Proceedings of the 9th International Conference on Network and Services Management (CNSM), 2013 [42]

- R. de O. Schmidt, R. Sadre, A. Sperotto and A. Pras, *Impact of Packet Sampling on Link Dimensioning.* Under review (TNSM).

Chapter 4: Pure Flow-based Link Dimensioning

Due to its wide presence in network devices, NetFlow measurements, or similar, have become an attractive source of information about the traffic. Although being a scalable solution for measuring traffic on high-speed links, the problem is that flow-level measurements only provide an overview of the actual traffic. In this chapter we describe an approach to create traffic time series out of flows, estimate traffic statistics needed by the dimensioning formula and, ultimately, estimate the required capacity. We show that this relatively simple approach is able to provide satisfactory results at larger timescales. This chapter extends the initial validation of the proposed flow-based approach as published in:

- R. de O. Schmidt, A. Sperotto, R. Sadre and A. Pras, *Towards Bandwidth Estimation using Flow-level Measurements.* In Proceedings of the 6th International Conference on Autonomous Infrastructure, Management and Security (AIMS), 2012 [45]

Chapter 5: Hybrid Flow-based Link Dimensioning

Motivated by the fact that the approach from Chapter 4 works for large timescales only, in this chapter we propose a more sophisticated approach to

estimate the required traffic statistics from flow measurements. The idea is to assume a parametrized model for packet arrivals within flows, and to use short-term packet captures for parameters tuning. Extensive numerical investigations show that this approach is able to accurately provide estimations of required capacity at very small timescales, such as 1ms. The content of this chapter has been published in:

- R. de O. Schmidt, R. Sadre, A. Sperotto, H. van den Berg and A. Pras, *A Hybrid Procedure for Efficient Link Dimensioning*. Computer Networks, 67, 252–269, 2014 [44]

Chapter 6: OpenFlow-based Link Dimensioning

Motivated by the increasing popularity and wide adoption of SDN-based technologies, in this chapter we propose an OpenFlow-based approach for link dimensioning. This approach uses messages defined by the OpenFlow protocol to retrieve traffic measurement data from the OpenFlow switch. Given that OpenFlow is able to, in theory, provide us with per-flow information (NetFlow/IPFIX style), the measured data from OpenFlow can be used as input to one of the flow-based approaches proposed in Chapter 4 and 5. The inaccuracy of measured data with current implementations of OpenFlow precluded the validation of the proposed OpenFlow-based approach. Nonetheless, we present a study on the quality of measured data obtained from a real OpenFlow switch and a virtual one using a widely adopted open source implementation of OpenFlow. The content of this chapter is partially based on the following publication:

- R. de O. Schmidt, L. Hendriks, A. Pras and R. van der Pol, *OpenFlow-based Link Dimensioning*. Demo at Innovating the Network for Data-Intensive Science Workshop (INDIS), ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2014 [37]

Chapter 7: Conclusions

In this section we summarize our main findings and contributions. We compare and discuss each of the proposed approaches in this thesis according to their respective ease-of-use and accuracy. We also indicate directions for potential future work.

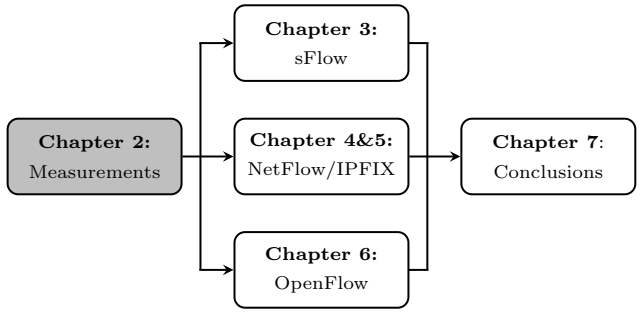
The last thing the men behind the curtain wants is a conscious informed public capable of critical thinking. Which is why a continually fraudulent zeitgeist is output via religion, the mass media, and the educational system. They seek to keep you in a distracted, naive bubble. And they are doing a damn good job of it.

— Zeitgeist, 2007

CHAPTER 2

Datasets and Traffic Characteristics

Traffic monitoring and measurements provide indispensable information for network operators to perform management actions in their networks. The goal of this chapter is to describe and characterize the traffic measurements used throughout the rest of this thesis for validating proposed link dimensioning approaches. Our dataset consists of hundreds of `pcap` *files with packet captures, which are made publicly available on SimpleWeb [104]. In this chapter we also present a comprehensive study on the (non-)Gaussianity property of traffic traces in our dataset, motivated by the requirement of Gaussian traffic from Equation* (1.2)*. Papers related to this chapter are [40, 41].*



The organization of this chapter is as follows:

- *Section 2.1 provides a brief overview on traffic monitoring and measurements.*
- *Section 2.2 describes how the packet measurements from our dataset are used throughout the thesis.*
- *Section 2.3 presents our measurements dataset.*
- *Section 2.4 thoroughly investigates sets out a thorough study on the Gaussianity fit of traffic of our dataset.*
- *Section 2.5 looks into the causes for bad fit of Gaussianity.*
- *Section 2.6 concludes this chapter.*

2.1 Measurements & Monitoring Overview

To begin with, it is important to clearly understand the difference between two terms often misused: *network traffic measurements* and *network monitoring*. From Wikipedia we learn that *network traffic measurements*[1] "is the process of measuring the amount and type of traffic on a particular network," accounting for what is seen. Also from Wikipedia, *network monitoring*[2] "is the use of a system that constantly monitors a network, notifying the administrator in case of outages."

Network traffic measurements can be done in two ways, *i.e.,* using *active* or *passive* techniques (or even a combination of both). Active techniques usually make use of tools, such as `Iperf`[3,4]. By injecting packets into the communication channel, these tools are able to measure, for example, throughput, packet loss and transmission delay. The downside of active measurements is, however, that it is a more intrusive measurement technique than the passive one.

In this thesis we focus on passive measurement techniques and tools. A widely used tool for passively measuring network traffic is the SNMP protocol [101]. SNMP use counters to provide a variety of basic statistics about the observed traffic. Alternatively, one can capture the observed packets and have more granular measurement data. There are many tools that allow for packet capturing, such as `tcpdump`[5] and `pf_ring` [88]. Aiming at scalability, one can use packet sampling tools, such as sFlow [62], to reduce the amount of captured packets.

Another way to passively measure traffic is by flow-level measurements. Among tools that measure flows, the most commonly found is Cisco's NetFlow [28]. Other vendors also implemented their own versions of NetFlow, for example, J-Flow from Juniper Networks [69]. In addition, there are many open source tools that perform flow-level measurements, such as YAF [61, 19] and Argus [98]. The recently proposed OpenFlow [83] protocol is also able to measure observed traffic on a flow basis, and the reported measured data by OpenFlow can assume a NetFlow-like form. The tools sFlow, NetFlow/IPFIX and OpenFlow will be discussed in more details in the next chapters.

[1] `http://en.wikipedia.org/wiki/Network_traffic_measurement`. Accessed on Jun. 2014.
[2] `http://en.wikipedia.org/wiki/Network_monitoring`. Accessed on Jun. 2014.
[3] `http://iperf.sourceforge.net/`. Accessed on Jun. 2014.
[4] `https://code.google.com/p/iperf/`. Accessed on Jun. 2014.
[5] `http://www.tcpdump.org/`. Accessed on Jun. 2014.

2.2 Converting Packet Captures

In this section we shortly describe the procedure we use to validate each of the link dimensioning approaches proposed in the following chapters. Our measurements dataset entirely consists of packet-level traffic captures (*i.e.,* `pcap`). By having the packet captures, we are able to validate the proposed dimensioning procedures against empirically defined ground-truth. We use of tools to convert this packet captures into measurements we want to use in our link dimensioning approaches. Figure 2.1 illustrates how we make use of our measurements dataset throughout this thesis.

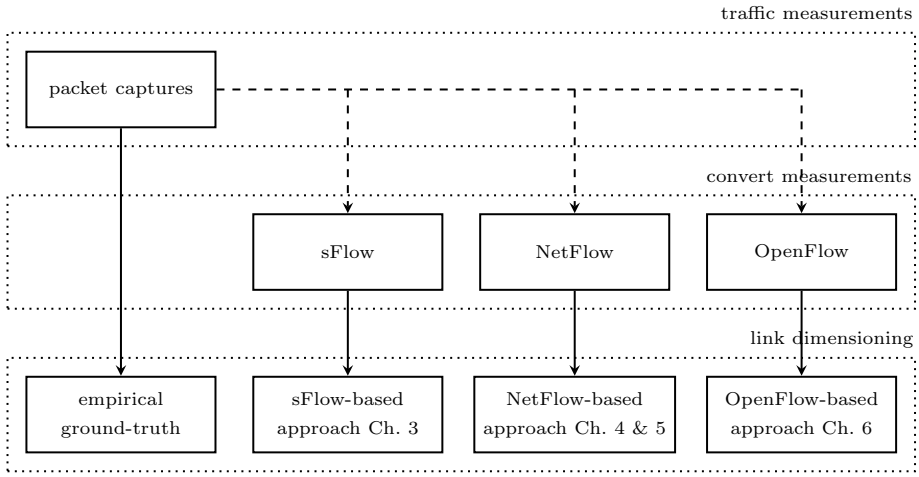


Figure 2.1: From traffic measurements to link dimensioning.

As one can see in Figure 2.1 we convert packet-level measurements to the ones used in the proposed approaches in this thesis. For each conversion we use a different tool. To convert from packets to sFlow sampled data, used by the approach of Chapter 3, we use our own implementation that emulates sFlow. Our implementation follows the sFlow definitions [95] as implemented by InMon's sFlow [62] and other sFlow tools such as `pmacct` [76]. The conversion from packets to NetFlow-like flows, used in Chapters 4 and 5, is done by the tool YAF [19]. Finally, in Chapter 6 we use real implementations of Open vSwitch[6] to collect traffic measurements produced by OpenFlow.

[6] `http://openvswitch.org/`. Accessed on Jun. 2014.

As above mentioned, the fact that packet-level measurements allow us to compute an empirically defined ground-truth and validate the results obtained from each of our proposed link dimensioning approaches. However, in real deployments the intermediate step to convert measurements should not be present. That is, the implemented link dimensioning approaches should receive as input measurements coming directly from the appropriate measurement tool.

2.3 Description of Measurements Datasets

In this section we describe the measurement dataset used to assess the Gaussianity of network traffic. The entire dataset comprises 768 15-minute traces, totaling 192 hours of captures. The trace duration of 15 minutes has been chosen in accordance with [110]. Longer time periods are generally not stationary due to the diurnal pattern. These traces come from different locations around the globe and account for a total of more than 18.5 billion packets. Traffic captures were done at the IP packet level, using tools such as `tcpdump`. Table 2.1 gives a summary of the data obtained from the six measurement locations. Note that the column "length" gives the total duration of the, not necessarily consecutive, 15-minute traces, i.e., a length of 1h corresponds to four traces. It is important to mention that no meaningful packet losses were observed for measurements directly performed by us (*i.e.,* locations A, B and C).

2.3.1 Measurement Locations

In this section we give a short description of the locations and time in which our measurements took place. Our dataset comprises traffic captures from six different locations. Three of them, namely A, B and C, are private university networks. While A consists of traffic from a link connecting a single education/research building in a university campus, locations B and C consist of traffic captures in the gateway of universities. Locations D, E and F consist of traffic from public backbone links. More details on each location is given next.

Location A

Location A is an aggregated link (2×1 Gb/s) connecting a university building in the Netherlands to the university's core router (university's gateway). Considering incoming and outgoing traffic, this link aggregates traffic from approximately 6500 hosts and has an average use of 15%. Most traffic in this link is actually internal to the university, i.e., from that building to other parts of the campus. Due to the small number of hosts, single activities, such as an overnight automatic backup, can drastically change the shape of the traffic. The

Table 2.1: Summary of measurements

abbr.	description	year	length	# of hosts	link capacity	avg. use
A	link from university's building to core router	2011	24h	6.5k	2 × 1 Gb/s	15%
B	core router of university in the Netherlands	2012	6h	886k	10 Gb/s	10%
C	core router of university in Brazil	2012	18h45m	10.5k	155 and 40 Mb/s	19%
D	backbone links connecting Chicago and Seattle	2011	4h	1.8M	2 × 10 Gb/s	8%
E	backbone links connecting San Jose and Los Angeles	2011–2012	5h	3M	2 × 10 Gb/s	10%
F	trans-Pacific backbone link	2012	13h15m	4M	n/a	n/a

measurement took place in a week day in September of 2011 with a duration of 24 hours. Therefore, this location comprises 96 successive 15-minute traces.

Location B

Location B is the 10 Gb/s up/down link at the core router of a university in the Netherlands. The link comprises all the incoming and outgoing traffic of the university. A total of approximately 886000 IP addresses were observed during the measured period and they generated an average link use of 10% (up to 15% in busiest hours). This is a full day measurement in which traffic was captured during the first 15 minutes of every full hour for a period of 24 hours. Therefore, this location comprises a total of 24 15-minute traces. The measurements of location A and B were made in the network of the same university. However, traffic patterns of these both are completely different. While one might say that $A \subset B$, actually not all the traffic from A is visible in B. That's because the former also comprise internal traffic to the very same building, which is not visible to the core router, *i.e.,* measurement point of B. In addition, B comprises traffic from the students residences in the university campus. This might result in a "higher than usual" volume of traffic during the night.

Location C

Location C is the core router of a university in Brazil. The aggregate of two links of 155 Mb/s and 40 Mb/s was measured during a week of November 2012. Each trace corresponds to the first 15 minutes of each full hour from 08:00 to 23:00 inclusive of every day during the measurement period. In this measurement it was observed an average use of 19% with around 10.5 thousand hosts mostly generating traffic related to web browsing, email and online services such as social networking and video streaming. Unlike the university from location B, the university of C does not have on-campus student residences and, therefore, traffic volume is expected to decrease considerably in the off-peak hours.

Locations D and E

The traces for location D and E are from CAIDA's public repository [17, 18]. Two unidirectional backbone links of 10 Gb/s each, from a Tier 1 ISP, were measured for each location. The original traces are captures of a full hour done on selected days. In location D, links interconnecting Chicago and Seattle (USA) were measured and the selected traces are from May and July 2011. In location E, links interconnecting Los Angeles and San Jose (USA) were measured and the selected traces are from December 2011 and January and February 2012. Each full hour of capture gives us 4 successive 15-minute traces. It is stated at CAIDA's web page that for one of the links from location D's pair, packet losses can be expected. For traces from location F, no information on packet loss is provided in the online repository.

Location F

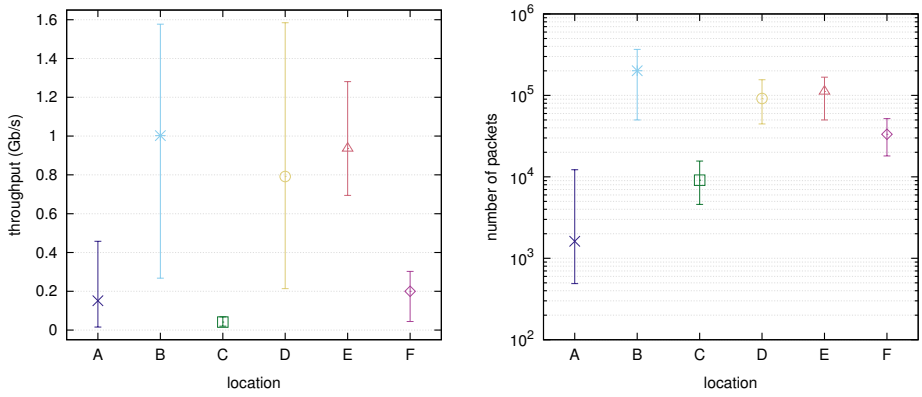
Location F is a transit link of the Widely Integrated Distributed Environment (WIDE)[7] to the upstream ISP. WIDE runs a major backbone of the japanese Internet. Measurements for this location come from the public MAWI repository [81]. The information on the link capacity as provided by MAWI on their website is not consistent with the throughput observed in the traces. Therefore, we cannot determine the average use of the link. These measurements consist of traffic captures from November 2012 to December 2012. In average, these traces aggregate traffic from more than 4 million hosts.

2.3.2 Traffic Characteristics

Table 2.1 presents the average link use for each location. Such value is not expected to be constant over the measurement period. Figure 2.2a shows the

[7] `http://www.wide.ad.jp/`. Accessed on Jun. 2014.

average, minimum and maximum traffic rate per 15-minute for each location. Locations with higher-capacity links are the ones in which traffic varies most. In case of 24-hour measurements from A and B, differences between minimum and maximum rates are due to traffic dissimilarities in diurnal and overnight periods.



(a) Average traffic rate

(b) Average number of packets

Figure 2.2: Traffic characteristics of the datasets from all locations. Error bars show minimum and maximum values for the respective statistics of each plot.

Figure 2.2b shows the average number of packets per 15-minute trace for each location. From this figure, one can infer, for each location, the average amount of packets after applying packet sampling techniques with different rates.

2.4 Overall Traffic Gaussianity Assessment

As mentioned in Section 1.4, one of the main requirements of the link dimensioning formula proposed in [109] is that traffic rates must be Gaussian distributed at the timescale of interest. This section is, therefore, dedicated to the study of such property in our measurements datasets.

In this section we first introduce the concept of Gaussianity and the methodology we use to check Gaussian fit of traffic traces, which was borrowed from previous works [71, 110]. Then we assess the Gaussian property of all traces from our datasets and relate their "degree of fit" to horizontal and vertical traffic aggregations. The former refers to the granularity of measurements (*i.e.,* timescale) and the latter refers to the amount of aggregated traffic sources (*e.g.,*

number of active hosts). Last, we present a study made with additional long-term measurements from location F (MAWI), where we assess the impact of traffic evolution from 2006 to 2012 on its Gaussian property.

2.4.1 Definition of Gaussianity

Let T be the timescale of traffic aggregation (*i.e.,* the same one to potentially be used in the link dimensioning formula as well), and let $L_1(T), \ldots, L_n(T)$ be the amount of traffic observed in time periods $1, 2, \ldots, n$ of length T. For any $T > 0$, we want to know if $L(T)$ is Gaussian distributed, *i.e.,* whether $L(T) \sim \text{Norm}(\rho T, \upsilon(T))$, where ρ is the average traffic throughput and $\upsilon(T)$ is the estimated variance of $L(T)$ given by, respectively

$$\rho = \frac{1}{nT} \sum_{i=1}^{n} L_i(T) \quad \text{and} \quad \upsilon(T) = \frac{1}{n-1} \sum_{i=1}^{n} (L_i(T) - \rho)^2 \, .$$

2.4.2 Assessing Traffic Gaussianity Fit

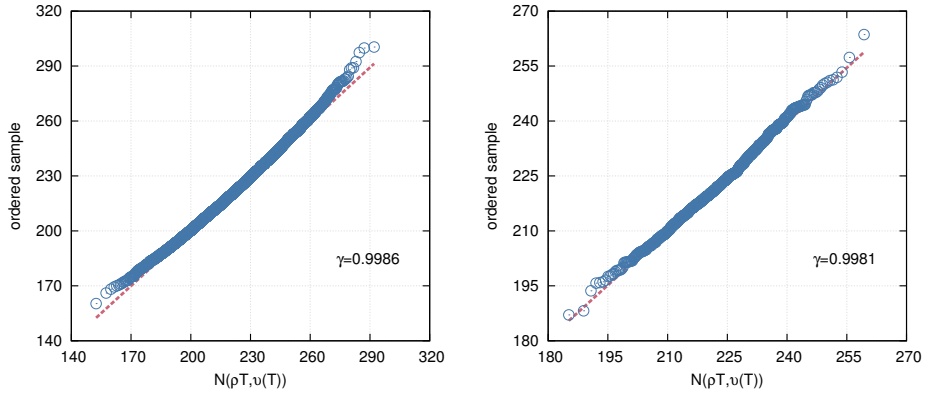
Quantile-quantile (Q-Q) plots can be used for a qualitative analysis of the Gaussian character of measured traffic. To create a Q-Q plot, the inverse of the normal cumulative distribution function $\text{Norm}(\rho T, \upsilon(T))$ must be plotted against the ordered statistics of the sampled data $L(t)$. Therefore, the pairs for a Q-Q plot are determined by:

$$\left(\Phi^{-1} \left(\frac{i}{n+1} \right), \alpha_{(i)} \right), \quad i = 1, 2, \ldots, n \quad , \tag{2.1}$$

where Φ^{-1} is the inverse of the normal cumulative distribution function, $\alpha_{(i)}$ are the ordered traffic averages for each time bin of length T and n the size of our sample (i.e., number of time bins of size T). Note that $\frac{i}{n+1}$ is used instead of $\frac{i}{n}$ because the 100th percentile is infinite for the normal distribution. However, for large sample sizes (i.e., large n), the difference is not significant [78, 79].

Figure 2.3 shows Q-Q plots generated from an example trace using two different values of T. For such plots, a traffic sample is considered "perfectly Gaussian" when all the points fall on the diagonal line. By visually analyzing the plots in Figure 2.3, one can conclude that, at both T, the traffic from the example trace is "fairly Gaussian", since only few points do deviate from the diagonal line.

When creating Q-Q plots of Internet traffic time series, it is common to see points at the high-end of the plot that fall distant from the diagonal line. This is due to the well known heavy-tail characteristic of traffic. This is a very important characteristic when the context of the study on Gaussianity is related



(a) $T = 100$ms (b) $T = 1$s

Figure 2.3: Q-Q plots for a single example trace at different T; this example trace is from location D.

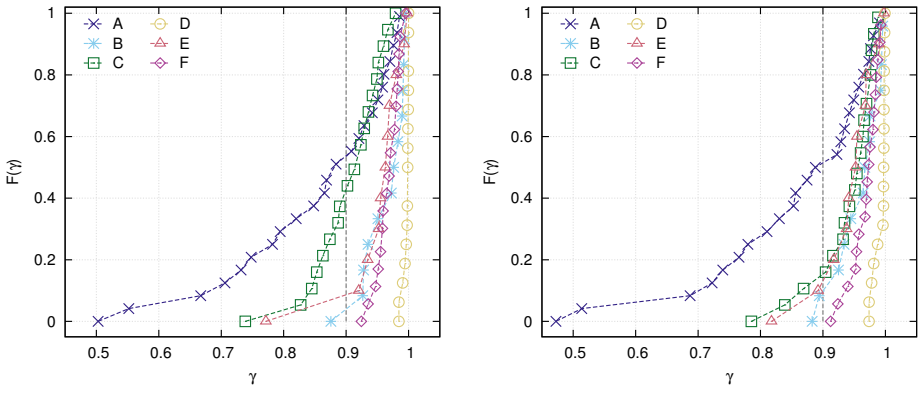
to management tasks such as bandwidth provisioning [96, 112] because such points represent significant fluctuations of traffic that occur at the considered timescale T. In the example of bandwidth provisioning, such fluctuations will impact traffic variance, which is an important parameter for computing the required link capacity for a given input traffic.

Q-Q plots provide a good visual analysis of the *goodness of fit* of the measured traffic compared to a Gaussian traffic model. However, a quantitative analysis is also needed to support observations from such plots. There are several procedures to quantify Gaussian *goodness of fit*. We opted for the *linear correlation coefficient* [15]. This choice was made to conform to the methodology followed by previous works [71, 110]. The *linear correlation coefficient* is defined by:

$$\gamma(x,y) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}} \quad , \tag{2.2}$$

where the pair (x, y) is the same as in Eq. (2.1).

Clearly, for a given traffic trace, $|\gamma| = 1$ if and only if all points lie perfectly on a straight line in the Q-Q plot. It is important to note that $\gamma \geq 0.9$ corresponds to a Kolmogorov-Smirnov test for normality at significance 0.05, which supports the hypothesis that the underlying distribution is normal. The values of γ for the example trace in Figure 2.3 are, respectively, $\gamma_{T=100ms} = 0.9986$ and $\gamma_{T=1s} = 0.9981$.



(a) $T = 100$ms (b) $T = 1$s

Figure 2.4: CDF of γ per location for all traces in our datasets; points are sampled for better visualization.

For a better understanding of goodness of fit for traces of our dataset within their respective locations, in Figure 2.4 we show the cumulative distribution function (CDF) of γ for all traces per location for arbitrarily chosen timescales of $T = 100$ms and $T = 1$s. At $T = 100$ms, around 56% of all traces have $\gamma \geq 0.9$, and at $T = 1$s it is around 83% of all traces. That is, at larger timescales most traces from our dataset are at least in the "fairly Gaussian" level. Clearly, the most problematic cases are traces from A and C, which comprise measurements from small networks and quiet periods of the network (*e.g.,* overnight). At $T = 100$ms around 47% and 59% of traces from locations A and C, respectively, have $\gamma \geq 0.9$. At $T = 1$s, the amount of Gaussian traces becomes 50% and 85% for A and C, respectively. The impact of lower number of active hosts and lower traffic average on the Gaussianity fit is further addressed in Section 2.4.4.

2.4.3 Horizontal Traffic Aggregation

The horizontal traffic aggregation is defined by the size of the time bin T. In this section we assess whether Gaussianity goodness of fit remains constant over various timescales. That is, we want to find out if a value of γ at a given timescale can give us an indication of how the traffic behaves at other smaller or larger values of T. According to [71], traffic tends to be more Gaussian-like at larger timescales and, therefore, larger horizontal aggregation of traffic is needed to justify Gaussian distribution. That's because isolated short term bursts, that would likely disturb Gaussianity fit, are smoothed

out due to averaging at larger timescales. Given that timescales from 5ms to 1s dominate the QoS as perceived by users, we have chosen to work with $T = \{0.001, 0.005, 0.01, 0.025, 0.05, 0.1, 0.5, 1\}$, all in seconds. Note that this is the same range of timescales used throughout this thesis, and it has also been used in previous works, such as [109].

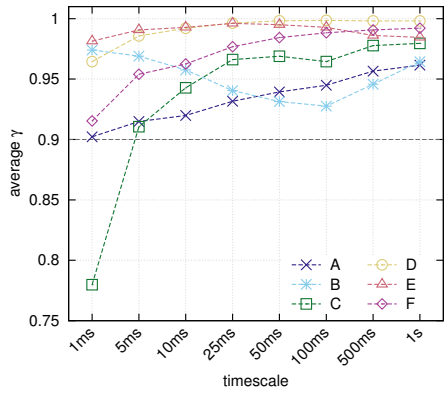


Figure 2.5: γ at various timescales for an example trace from each location.

Our analysis of the impact of vertical aggregation of traffic on its Gaussian fit starts by a randomly selected example trace from each location. Each line of the plot in Figure 2.5 presents the Gaussianity goodness of fit γ calculated for each one of the example traces for all chosen timescales. From this plot it is clear that, for these traces, traffic becomes less Gaussian on shorter timescales mainly when the link aggregate is not too large in terms of transferred data, which is the case for locations A, C and F. That is, the shorter T, the less traffic is aggregated per time bin and the higher the traffic variance due to bursts typically caused by individual hosts. A study on the impact of individual hosts on the Gaussian fit of traffic is latter presented in Section 2.5.

According to [71], a very short T, *i.e.,* lower than milliseconds, might be too close to the packets' transmission intervals. That would result in a time series with a binary behavior, where we may have or have not packets being transmitted within the period of length T (*i.e.,* ON/OFF behavior). Such a time series is, obviously, not Gaussian. This can be even more problematic if we consider links that aggregate traffic of very few hosts, because we may have binary-like traffic time series even in the milliseconds timescale. For example, the example trace from location C shows a bad Gaussianity fit at $T = 1$ms. This problem is alleviated when increasing T over a certain threshold, where γ becomes fairly stable. The same behavior, but with less impact on γ, can be

observed for traces from other locations. Interestingly, the example trace from location B has a different behavior than others in Figure 2.5 because its goodness of fit γ actually improves at smaller T. However, the observed fluctuation of γ in this case is not large: γ is greater than 0.9 for all values of T. A similar situation has also been observed in [110].

An important take away of the analysis in Figure 2.5 is that, on the one hand, it would not be completely safe for an arbitrary location to assume Gaussianity at very short timescales solely based on the fact that the traffic is Gaussian at a larger timescale. That is, if the traffic is Gaussian at $T = 1$s, it does not necessarily mean that the same traffic will remain Gaussian at $T = 1$ms. On the other hand, Figure 2.5 also indicates that γ, to a wide extent, monotonously increases with T and, hence, one can assume Gaussianity at timescale T_1 for a particular traffic if the same traffic is Gaussian at T_0, and even more safely if it is also Gaussian at T_2, where $T_0 < T_1 < T_2$.

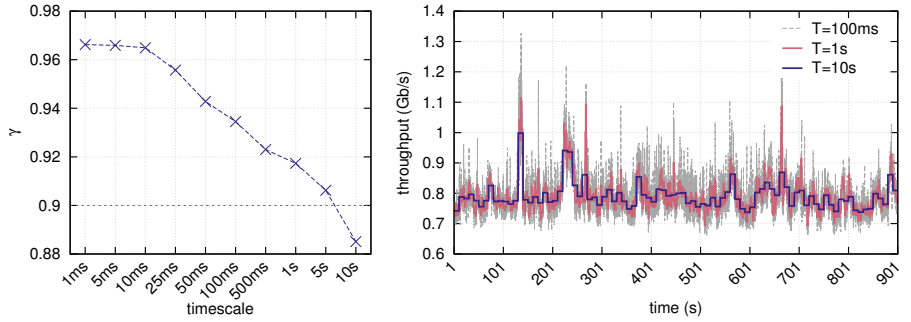


Figure 2.6: Example of decreasing γ with increasing T using a trace from location E (left); and the traffic time series of the same trace at different T (right).

The averaging of traffic bursts at larger timescales might not improve Gaussianity when these bursts belong to consecutive time bins or at least happen very close to each other. Figure 2.6 shows an example in which increasing the size of T does not necessarily increase the Gaussian fit of the traffic aggregate. In this figure we have extended the range of timescales up to 10s for matters of illustration. One can see that at $T = 10$s traffic is not Gaussian-distributed anymore, while at very small timescales it has a very good γ. The time series of this trace, which is an example from location E, clearly shows the problem. In the period between 120s and 270s there are many traffic bursts that when aggregated at larger timescales, such as 1s or 10s, are averaged together, creating huge traffic bursts that differ too much from the baseline traffic. As a consequence the Gaussian fit of traffic is decreased. Note that we expect that

at very large timescales, *e.g.,* minutes, the Gaussianity fit is lower. That is because daily traffic patterns might become very distinguishable in the time series, resulting in a binary-like behavior, similar to what is found at very small T.

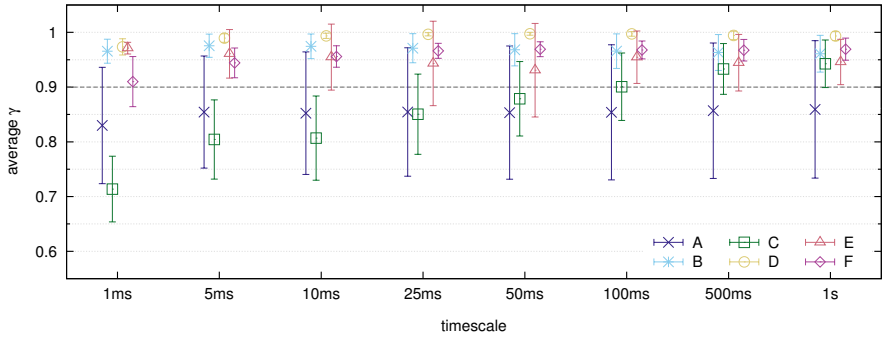


Figure 2.7: Average γ at various timescales for all traces in our dataset.

To validate the relationship between horizontal traffic aggregation and Gaussianity fit, we have calculated the goodness of fit γ for all traces in our entire dataset at various timescales. Figure 2.7 shows the average γ, and the respective standard deviation, for all traces per location at the timescales of interest. The idea is to show that, in general, traffic for a given location remains Gaussian through different timescales. We can observe that for all locations, γ either increases at larger timescales or remains almost constant. Certainly the most interesting case is the one of location C where, in general, traffic is not Gaussian at smaller timescales but it becomes Gaussian at larger timescales. This is likely due to the low traffic averages of C that, consequently, at small T do not aggregate enough traffic to justify Gaussian assumption. Moreover, again this complies with statements from [71] on how Gaussianity should increase with T. For the case of location A, however, traffic is mostly not Gaussian at any of the considered timescales. Since A is a 24-hour measurement and it has a quite low number of active hosts, one possible explanation is that mainly during the overnight period traffic is very unsteady, *i.e.,* strongly non-stationary and, hence, non-Gaussian.

Finally, it is also interesting to know the consistency of γ of a location over all considered timescales. Recall that Figure 2.5 shows γ for a set of example traces and Figure 2.7 shows the average for all traces from a location at specific timescales. One way to find out the stability of γ for each location is to compute, individually for each trace, the standard deviation of the trace's γ at various

timescales. This metric was proposed in [110]. Hence, for a trace with gamma γ_T for $T = 1\text{ms} \ldots, 1\text{s}$, we compute

$$\sigma_\gamma = \sqrt{\text{Var}[\gamma_T]}\ . \tag{2.3}$$

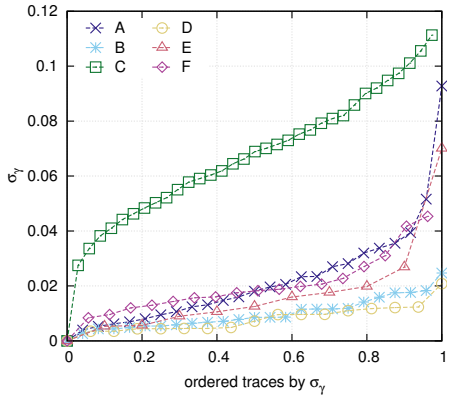


Figure 2.8: Variance of γ across various timescales; points sampled for visualization.

Figure 2.8 shows the results for each location with the traces sorted left-right by their standard deviation σ_γ. The lines reveal that for more than 50% of all traces $\sigma_\gamma \leq 0.05$, and that for about 90% of all traces $\sigma_\gamma \leq 0.09$. Analyzing each location separately, we see that $\sigma_\gamma < 0.02$ for almost all traces from B and D, and for more than 95% of traces from A, E and F, $\sigma_\gamma < 0.04$. For all traces, except some from location C, $\sigma_\gamma < 0.1$. Only for around 5% of traces from C $\sigma_\gamma \geq 0.1$. These results strengthen our previous conclusions that Gaussianity is quite constant across timescales, and that traffic exhibiting good Gaussian fit at T_0 and T_2 is likely to be Gaussian also at T_1.

2.4.4 Vertical Traffic Aggregation

Previous works [71, 110] have also studied the impact of vertical aggregation on the Gaussianity of traffic. Vertical aggregation refers to the amount of aggregated traffic sources. An important question is how many sources are needed to guarantee the Gaussian characteristic of the traffic. Furthermore, we are interested in a definition of traffic source that can be easily used to calculate the number of active sources. For example, a traffic source is not necessarily equivalent to a TCP connection. In [110] the authors attempted to quantify the number of hosts (measured as number of observed IP addresses) necessary to

justify the Gaussianity assumption. To do so, they sampled traffic from randomly selected hosts and compute γ for it. They conclude that few dozens of hosts would be enough to justify traffic Gaussianity.

We believe that it can be risky to solely rely on the number of observed hosts since this assumes that all hosts behave uniformly in all networks. It is not clear whether the same number of hosts sufficient for Gaussianity in network X would also be sufficient in network Y. For example, hosts in a university campus network may behave completely differently from the sources observed in a backbone link. Later in Section 2.5, we show that depending on the network, the behavior of individual hosts might actually have a negative impact on Gaussianity fit. An alternative approach is to relate the level of vertical aggregation to the amount of traffic aggregated. However, this can be also dangerous since individual hosts can also have high transmission speeds, as already observed in [71]. Therefore, we study in the following the impact of vertical aggregation on Gaussianity both in terms of (i) the number of hosts and (ii) the amount of traffic aggregated.

For this analysis we have used three measurements, two already part of the initially introduced dataset in Section 2.3, namely locations A and B, and one extra measurement from location C, which we henceforth refer as C_2 (note that measurements of C_2 aggregate traffic from less users than C because the former took place during students vacations). All three measurements in this analysis are a 24-hour traffic measurement. The difference is, however, that while A comprises 24 hours of uninterrupted packet capturing, the other two consist of captures of the first 15 minutes of each hour. The natural diurnal pattern present in such measurements results in strong variations in the network usage. In addition, hosts that are active overnight often behave quite differently from those active during the daytime. This allows us to study the impact of wide range of scenarios on the Gaussianity.

Remark: for the following analysis we show results in Figures 2.9, 2.10, and 2.11. The size of the x-axis is defined by the number of traces, which is given in the figure's caption. Each trace corresponds to a tic in the x-axis. Goodness of fit γ and traffic averages were calculated using bins of size $T = 1$s. This remark is also valid for Figure 2.12 presented next in Section 2.4.5.

The top plot of Figure 2.9 shows γ for each 15-minute trace of location A. One can see that γ oscillates a lot across the measurement period. Good Gaussianity fit is found even during the overnight period, what would not be expected considering the small number of active users in the network (bottom plot). However, as shown in the center plot, between 23:00 and 01:00, there is an increase on the traffic average which seems to be the reason of the good fit. A smaller increase can also be observed from 02:00 to 04:00. This might have been the result of automatic operations, such as overnight backups. The

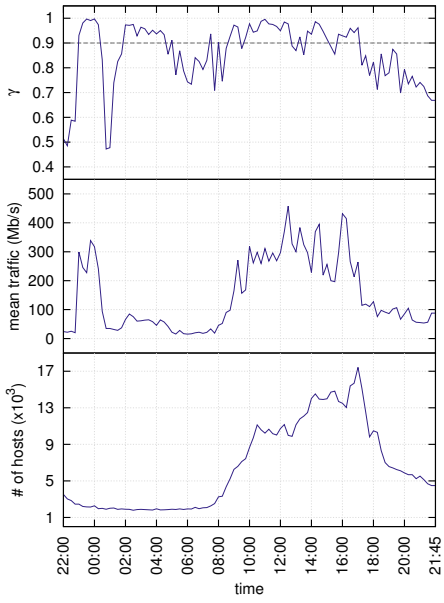


Figure 2.9: Goodness of fit γ (top), mean traffic rate (center) and number of hosts (bottom), at $T = 1$s, for all traces from A (96 traces).

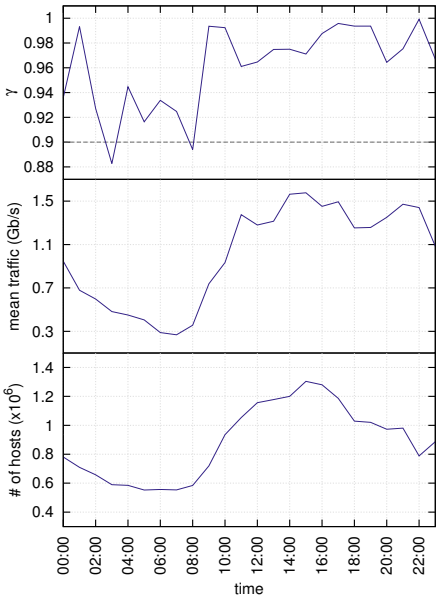


Figure 2.10: Goodness of fit γ (top), mean traffic rate (center) and number of hosts (bottom), at $T = 1$s, for all traces from B (24 traces).

figure also shows that traffic averages during the day are generally higher due to the higher number of active users in the network. Consequently, Gaussianity of traffic tends to be more regular in the busiest period of the monitored link. The results indicate that Gaussianity depends more on the behavior of hosts than on the quantity of active hosts. Although we have more than thousand active hosts at 01:00 and 21:45, the Gaussianity fit is low. Furthermore, Figure 2.9 shows that a high traffic rate can be a better indicator for good Gaussianity than the number of users. Note that the opposite is not necessarily true.

Location B has many more hosts than location A, as seen in the bottom plot of Figure 2.10. One of the reasons is that this link also transports traffic from the residential buildings located on the university campus and the public servers. As shown in the top plot of Figure 2.10, only the traces of 03:00 and 08:00 do not have a good enough Gaussianity fit of traffic, *i.e.,* $\gamma < 0.9$. In these moments, traffic averages and number of active hosts were quite low compared

to other periods of the day. Although one cannot argue that Gaussianity is as bad as observed for the overnight period in A (Figure 2.9), it is clear that it is unstable between 00:00 and 09:00. Again it seems that the behavior of a few users determine the Gaussian characteristic during the light-loaded period of the link and, once again, we observe that the high traffic rate is a better indicator for a high γ value than the number of users. For example, γ closely follows the traffic rate in the period from 19:00 to 23:00. The observation that the opposite is not necessarily true remains valid.

Figure 2.11 shows the goodness of fit γ for the additional measurement C_2. As one can see in the top plot of this figure, although the values of γ are not very high (*i.e.,* close to 0.9), they are fairly stable. Indeed, it seems that it is affected neither by the traffic rate nor by the number of hosts. We believe that, due to the period of vacations, the traffic characteristics mainly arise from the rather constant behavior of the university's employees and automated processes, while variations caused by, for example, file transfers are rare. The main take away from Figure 2.11 is that with links that have low capacity or low activity the overall host behavior becomes the dominating factor.

2.4.5 Long-term Traffic Evolution

The goal of the study presented in this section is to assess whether long-term traffic evolution has an effect on the Gaussian property of traffic. While one expects that the increasing traffic rates in the past years would actually improve Gaussianity fit, the motivation behind this study is that traffic evolution would potentially be caused by recently emerged applications that have very distinct behavior. One the one hand, services such as Facebook would be responsible for many connections with few transferred data (*i.e.,* short flows) [52]. On the other hand, online video streaming (*e.g.,* YouTube) and cloud storage (*e.g.,* Dropbox) services would be responsible for connections with, generally, large amount of transferred data [48, 52].

To do so, we use an additional measurement from location F, which we name F_2. This extra dataset comprises 178 15-minute long traces dating from August 2006 to December 2012, totaling more than 44 hours of packet captures. Our choice for starting with measurements from 2006 is because, to the best of our knowledge, the last scientific publication that addressed the Gaussianity fit of traffic aggregates was [110] in 2006. Traces from F_2 have an average of 1 million hosts. Note that traces dating from 2011 or older have a much lower average number of hosts and some of those from 2012 have a much higher number of hosts (bottom plot in Figure 2.12). For unknown reasons, the measured link experienced several moments of huge peaks on the number of hosts transferring data from September 2011 to December 2012. However, these peaks on number

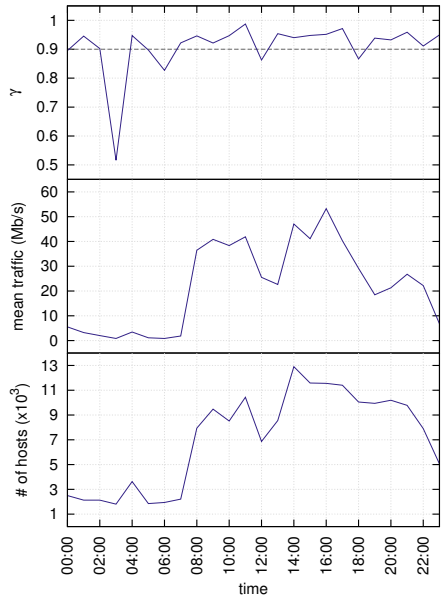


Figure 2.11: Goodness of fit γ (top), mean traffic rate (center) and number of hosts (bottom), at $T = 1$s, for traces of an extra 24-hour measurement period from C_2 (24 traces).

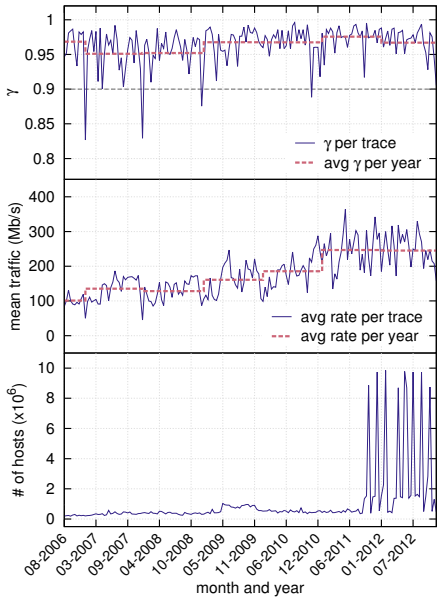


Figure 2.12: Goodness of fit γ (top), mean traffic rate (center) and number of hosts (bottom), at $T = 1$s, for traces of an extra long-term measurement period from F_2 (178 traces).

of hosts did not result in abnormal traffic rates, as seen in the center plot of in Figure 2.12. Unfortunately, MAWI does not provide additional details of such measurements that could potentially lead us to a better understanding on the causes for these differences in the number of hosts.

The top plot in Figure 2.12 shows the Gaussian goodness of fit γ calculated for all traces from F_2. We observe that the (already good) average goodness of fit has only slightly increased from 2006 to 2012, while traffic throughput has more than doubled. It should be noted, though, that the increased throughput results in less variation of γ. The number of hosts has not changed significantly during the measurement period. The peaks on the number of hosts in traces from end of 2011 and from 2012 did not result in worse or better Gaussianity fit.

2.5 Causes of Bad Gaussian Fit

In the previous section we have assessed the Gaussianity "degree" for the whole measurement dataset used in this thesis. However, considering the adopted dimensioning formula from [109], Gaussianity fit is of paramount importance for the problem of link dimensioning in the context of this thesis. Therefore, we believe that it is also important to understand the reasons for traffic being not Gaussian-distributed. Our assumptions are that: (1) Gaussianity is mostly disrupted by traffic bursts that are much higher than the traffic average; (2) these bursts are usually generated by one or very few applications; and (3) there are very few hosts creating those traffic bursts.

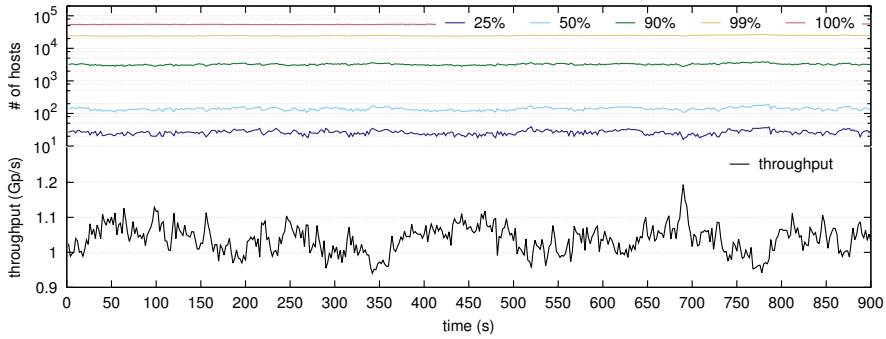
Therefore, in this section we study the relationship between traffic properties and hosts behavior and the Gaussianity fit of traffic. We use traces from three locations of our whole dataset, namely, B, C and E. First, we study the impact of traffic bursts on the degree of Gaussianity fit. Then, we identify traffic bursts and analyze the applications behind them. Finally, we assess the impact of individual hosts behavior on Gaussianity.

2.5.1 Impact of Bursts on Gaussianity

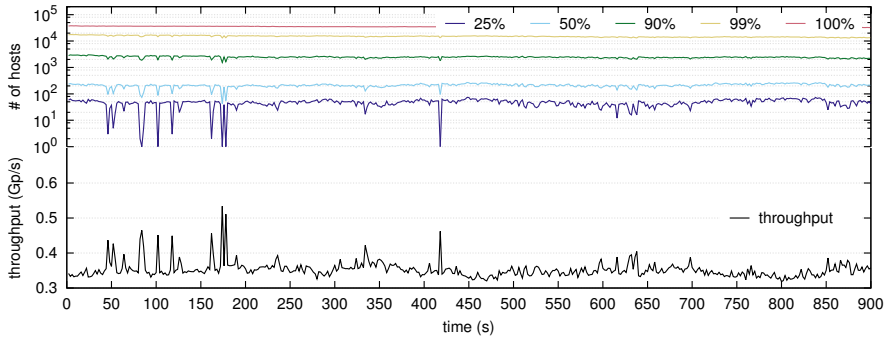
Normal distributed traffic (Gaussian) is expected to have bursty and calm moments. By definition, if the traffic aggregate $L_i(T)$ follows a normal distribution $\text{Norm}(\rho T, \upsilon(T))$, the probability that it exceeds a threshold x is given by the complementary CDF

$$P(L_i(T) > x) = 1 - \frac{1}{2}\left(1 + \text{erf}\left(\frac{x-\rho}{\sqrt{2\sigma^2}}\right)\right) . \tag{2.4}$$

Figure 2.13 shows the difference between two traces from location E with low and high Gaussianity fit, respectively. The curve in the bottom half of these plots shows the traffic aggregate of the sample traces over the measurement period of 15 minutes. Note that in this figure we have chosen $T = 2$s for visualization purposes, while for the following experiments in this and next subsections, we use $T = 100$ms and $T = 1$s. The trace of Figure 2.13a has $\gamma = 0.9977$, which is one of the highest Gaussianity fir among all traces of our dataset. One can clearly see that traffic of this trace has regular ups and downs and in any moment a burst really protrudes from the baseline traffic. Contrariwise, the trace of Figure 2.13b has $\gamma = 0.8175$, which is the lowest goodness of fit for traces from location E. In this case, one can easily notice the very high bursts during the time period 50–200s and at time 420s. (The analysis of the traffic shares presented in the top half of plots of Figure 2.13 is done in Section 2.5.3).



(a) Good Gaussian trace ($\gamma = 0.9977$)



(b) Bad Gaussian trace ($\gamma = 0.8175$)

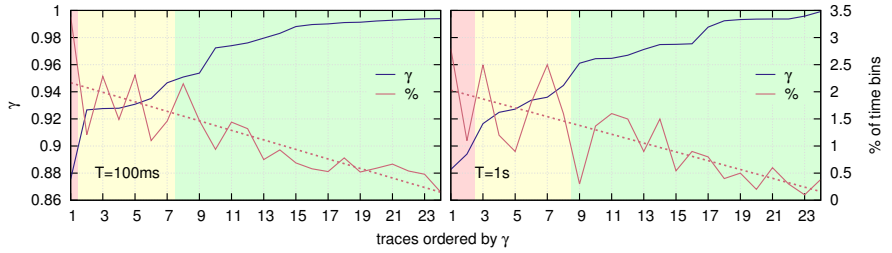
Figure 2.13: Traffic aggregate (bottom half) and traffic shares (top half) at $T = 2$s for two sample traces from location E.

We have manually inspected and compared several traces with poor and good Gaussianity and noticed that such bursts are typical for poorly Gaussian traces. In order to assess this behavior systematically, we define a burst as a time bin where the traffic aggregate exceeds the threshold θ defined by

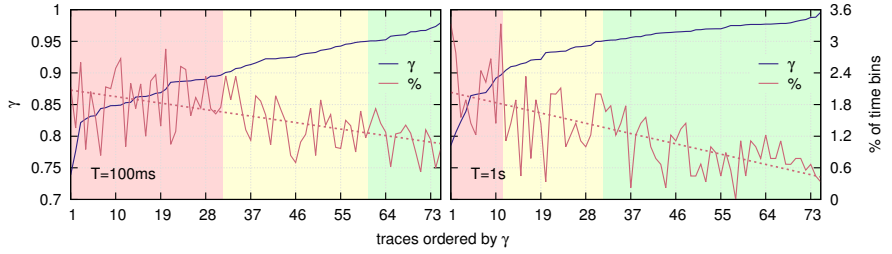
$$\theta = \rho + 3\sigma \ . \tag{2.5}$$

That is, θ is three standard deviations above the trace average rate ρ. A similar definition of burstiness has been used in [100, 73]. According to Eq. (2.4) this should only happen with probability 0.00135 in perfect Gaussian traffic.

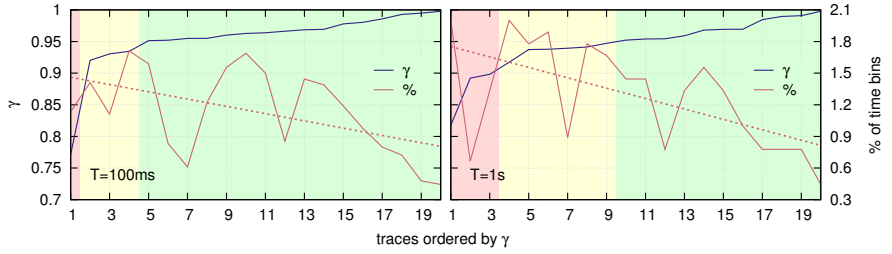
The plots of Figure 2.14 show for each trace of different locations its Gaussian fit γ and the percentage of time bins that exceed the above threshold θ, for $T = 100$ms (left) and $T = 1$s (right). In these plots traces are sorted left-right



(a) Traces from B



(b) Traces from C



(c) Traces from E

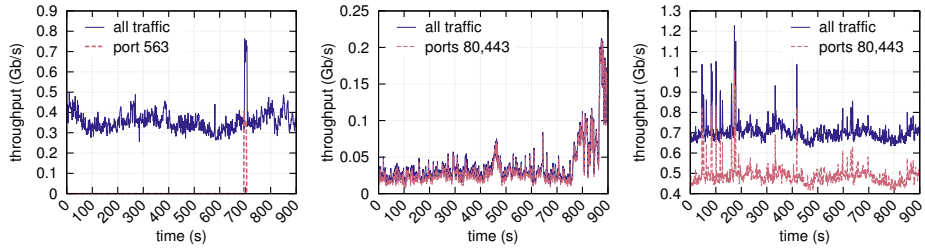
Figure 2.14: Percentage of time bins with bursts at $T = 100$ms (left) and at $T = 1$s (right).

by their Gaussianity fit, *i.e.*, trace 1 (left on the x-axis) is the trace with the lowest γ. No that the traces positioning in the x-axis varies from $T = 100$ms to $T = 1$s due to their different values of γ at different timescales. For example, trace 1 in the left side plot of Figure 2.14a may not be the same as trace 1 in the right side plot of the same figure. Moreover, the colors of the background in these plots indicate the regions of different γ values: the red means traces have $\gamma < 0.9$; the yellow means $\gamma \geq 0.9$ and $\gamma < 0.95$; and green means $\gamma \geq 0.95$. These considerations are also valid for plots from Figure 2.16 to 2.17.

Although the resulting curves in these plots depict strong fluctuations independently of T, we observe an inverse relationship between the amount of bursts exceeding the threshold and the Gaussian fit. That is, non-Gaussian traces tend to have more bursts than Gaussian ones. This tendency is highlighted by the least-squares-fitted diagonal dotted line in the plots of Figure 2.14. Note that this is not a trivial outcome since non-Gaussianity could be caused by the absence of bursts as well. In fact, a few non-Gaussian traces have a very small number of bursts.

2.5.2 Impact of Applications on Gaussianity

In the previous section we have shown the relationship between bursts and (non-)Gaussianity. In this section, we study the impact of traffic bursts from certain applications on the Gaussianity fit. Note that we use the straightforward port-matching method for identifying applications. However, we are aware of the drawbacks of such method. Challenges related to traffic classification and its connection to Gaussianity are further discussed in Section 2.5.4.



(a) Trace from B, $\gamma = 0.8940$ (b) Trace from C, $\gamma = 0.7853$ (c) Trace from E, $\gamma = 0.8175$

Figure 2.15: Traffic aggregates at $T = 1$s and the port causing the bursts.

Figure 2.15 shows the traffic aggregates of three sample traces, from different locations, with a low Gaussianity fit. The upper curve gives the aggregate of all traffic. We observe several bursts. For the trace from location B, the lower curve only shows the aggregate for traffic transferred on port 563, *i.e.,* Network News Transfer Protocol (NNTP). For the sample traces from locations C and E, the lower curve shows the aggregate of traffic transferred on ports 80 and 443 (*i.e.,* HTTP and HTTPS, respectively). It can be seen that the protocol-specific curves follow closely the shape of the bursts for all the three examples. In fact, we have observed that typically a burst consists entirely of traffic from only one application and, hence, removing the specific traffic of such application from the

trace would also remove the bursts. However, it is important to consider that all bursts in a trace might not necessarily be caused by the same application.

In order to validate this observation for the entire dataset, we have calculated for each burst that exceeds θ (as defined in Eq. (2.5)) the share of the traffic on the most active port in the time bin of that burst, and computed an average share for all bursts of a trace. The plots in Figure 2.16 show the resulting (average) traffic share for $T = 100$ms (left) and $T = 1$s (right). Again, traces are sorted on the x-axis by their respective Gaussianity fit γ and the background color indicates their "degree of fit" in the same way as in Figure 2.14. We observe that for traces with low γ, the traffic bursts that exceed θ mainly consist of traffic from the most active ports, and that this relationship weakens with increasing γ. Note that the share never reaches 100%. Clearly, this is because the time bin containing the burst also contains normal baseline traffic. Furthermore, we observe a generally high share for all traces of location C. This is because HTTP(S) is the most dominant traffic at this location (as one can see in the example trace of Figure 2.15b).

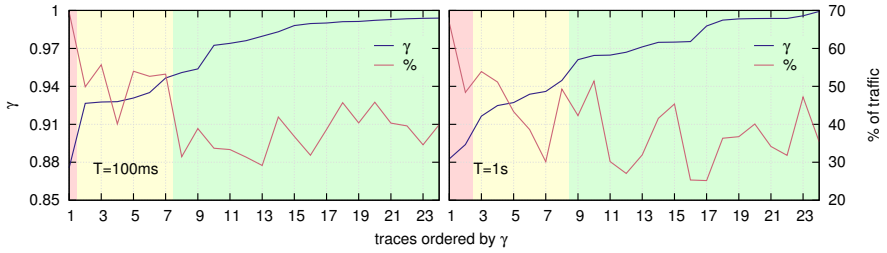
2.5.3 Impact of Individual Hosts On Gaussianity

The previous analysis has shown that bursts are mostly caused by single applications. In this section we investigate how individual hosts contribute to the traffic in such bursts. The top half of the plots in Figure 2.13 show the absolute number of most active hosts that are responsible for 25%, 50%, 90%, 99% and 100% of the traffic sent in a given time bin. More formally, let $b_1(t) \geq b_2(t) \geq \ldots$ be the sorted number of bytes sent by the hosts in the time bin t. That is, $b_1(t)$ is the number of bytes sent by the most active host in the time bin t, $b_2(t)$ is the number of bytes sent by the second most active host in t, and so forth. The number $q_s(t)$ of the most active hosts responsible for a share s of the traffic in time bin t is defined as

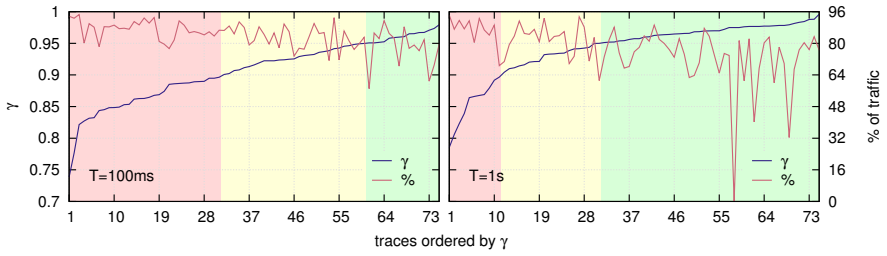
$$q_s(t) = \min_{\sum_{i=1}^{x} b_i \geq s \cdot B(t)} x, \tag{2.6}$$

where $B(t)$ is the total number of bytes sent in the time bin t.

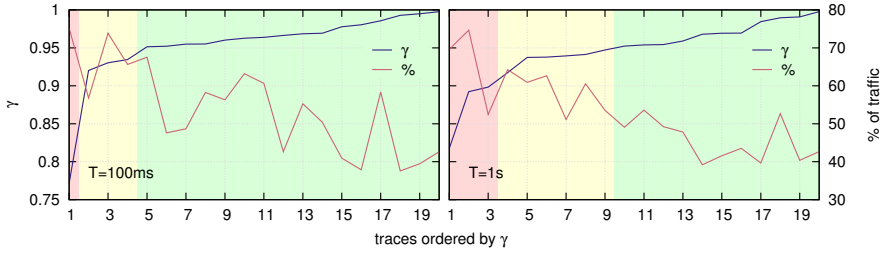
One can see that while for the good Gaussian trace in Figure 2.13a the number of hosts that are responsible for any share of the traffic remains quite constant over time. In any moment the number of contributing hosts drops considerably, not even during the highest traffic burst of this example trace, around 690s. On the contrary, for the trace with bad Gaussianity fit in Figure 2.13b, the number of hosts that contribute to a certain share of traffic significantly drops during bursts. For example, during the burst at time 420s, only one host sends 25% of the traffic, which more or less corresponds to the difference between the 0.45 Gb/s peak throughput of the burst and the baseline throughput



(a) Traces from B



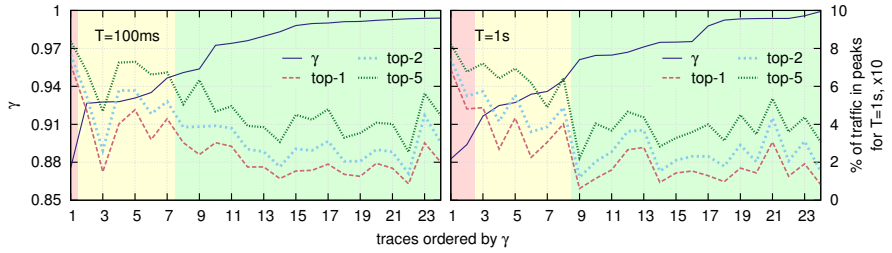
(b) Traces from C



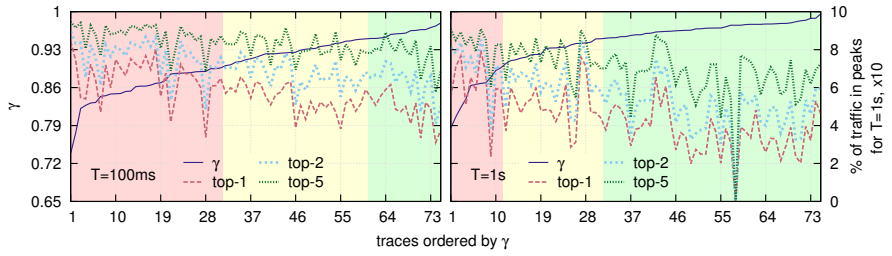
(c) Traces from E

Figure 2.16: Share of the most active applications in bursts at $T = 100$ms (left) and at $T = 1$s (right).

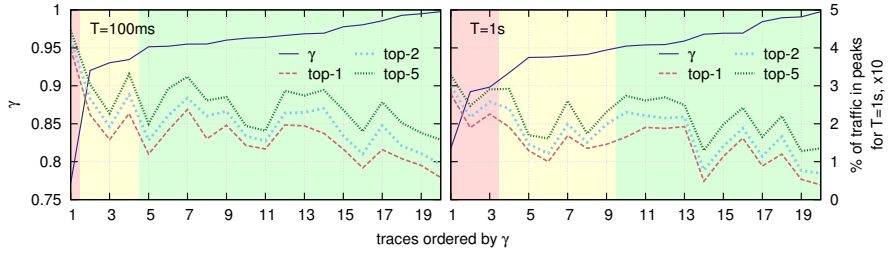
of 0.35 Gb/s, *i.e.*, that burst is caused by traffic from one single host. Outside the bursts, a much larger number of hosts contribute to the 25% traffic share. In general, the non-bursty part of the traffic is in accordance with the observations made in [14] that typically 90–95% of IP traffic is generated by 10–5% of the sources. The authors of [14] also found bursts in their traffic, but mostly connected them to attacks. However, after a manual inspection of several bursts, we consider it unlikely that the bursts in our traces are caused by malicious



(a) Traces from B



(b) Traces from C



(c) Traces from E

Figure 2.17: Share of traffic for top-hosts (IPs) in bursts at $T = 100$ms (plots) and at $T = 1$s (plots).

activities. The size of the bursts also makes them very unlikely to be caused by source-level bursts on packet level [66].

Once again we have validated this observation for all traces from locations B, C and E. We have calculated for each burst higher than θ the share of the most active host, the two most active hosts, and the five most active hosts to the traffic in that burst and computed the average shares for all bursts of a trace. The plots in Figure 2.17 show the resulting traffic shares for each trace per location at $T = 100$ms (left) and at $T = 1$s (right). We observe for all traces

that, independently of the Gaussianity fit, very few hosts are responsible for a significant amount of transferred traffic during the bursts. In some situations, at $T = 1$s, less than 5 hosts are responsible for more than 80% of all burst traffic in traces from location B and C and more than 35% in traces of E.

2.5.4 Challenges on Traffic Classification

In this section we have identified applications to the level of protocol by matching the port numbers (*e.g.,* 80 and 443 to HTTP and HTTPS, respectively). However, it is known that a plethora of applications are currently running on top of HTTP(S) and identifying those applications is not a straightforward task. The authors of [36, 107] point out that many applications do not have IANA registered ports. Instead, they make use of well-known ports or tunneling to prevent detection and deceive filtering or firewalls. For instance, BitTorrent can also make use of random ports, complicating their identification by default communication ports.

Sophisticated traffic classification almost precludes application identification with goals of Gaussianity assessment. That is, to ultimately have a sort of *rule of thumb* that would allow us to make assumptions on the degree of Gaussianity of a given traffic aggregate based on the mix of applications found within it seems to be more complex than simply measuring the traffic for a short period and performing the same operations as we have done in this paper (*i.e.,* computing Gaussianity goodness of fit γ). Nonetheless, we have shown that bursts of traffic tend to belong to a handful of hosts and being transferred on a very limited range of ports, even for large networks. Hence, we also see the work presented in this section as a first contribution toward an application-oriented method to assess the Gaussianity assumption: instead of performing a costly network-wide measurement on packet-level, researchers or network operators would first identify hosts that contribute most to bursts in the main traffic, and later they would explore further the applications that are being used by those hosts (*i.e.,* to a higher level than the application protocol that they use).

2.6 Concluding Remarks

The datasets presented in this chapter comprise extensive measurements from four continents and covers diverse scenarios, from small campus networks to 10 Gb/s backbone links. These datasets are composed by recent packet captures, which were done without applying any sort of sampling. These datasets are later used in the validation of the proposed link dimensioning approaches in this thesis.

We have verified the assumption of Gaussianity fit of our entire dataset. Our results show that the assumption of Gaussianity still holds for current network traffic, indicating that the evolution of the Internet in the past years has not had a significant impact on its Gaussian characteristics. Indeed, most of the analyzed measurement locations show a high or very high overall degree of Gaussianity for a wide range of considered aggregation timescale. However, this degree can vary depending on the level of vertical aggregation and is usually highest during the busiest period of the network, *i.e.,* during daytime. Our findings also suggests that it is safer to relate the degree of Gaussianity to traffic throughput than to the number of active hosts for high-speed links. The number of active hosts is less reliable as indicator for Gaussianity because hosts from different networks may behave differently. Furthermore, we have illustrated the invariance of the Gaussianity property by our study of a trans-Pacific backbone link over a period of six years. Although the amount of traffic transported by that link has considerably changed during the measurement period, the degree of Gaussianity has nearly stayed constant. Therefore, we can conclude that the mathematical models that build up the link dimensioning formula from [109] are still valid, especially for the, from the viewpoint of network operators, most interesting periods of high network activity.

We have shown that the degree of Gaussianity of network traffic is directly linked to the presence of extreme traffic bursts. While fairly Gaussian network traffic is mostly burst free, traffic with a low Gaussian fit is up to 3.6% of its duration bursty. In addition, we have shown that these bursts are mostly created by single applications. In particular, traffic bursts at two of our measurement locations mostly consist of HTTP(S) traffic. We have also observed that bursts in traffic with a low Gaussian fit tend to consist of traffic from only one application. Finally, we have shown that the traffic inside bursts is sent from only a few hosts. Our results enable the conclusion that poor Gaussianity is caused by short but intensive activities of single network hosts. This suggests that the bursts are related to the transfers of big files using unlimited access speeds over fast links. Our findings confirm that the concept of alpha and beta traffic, introduced by [100] in 2001, is still valid for recent network traffic. However, it is worth to note that two of our measurement locations, namely B and C, are university core routers that connect a rather homogeneous set of hosts with identical, or at least similar, link speeds to the Internet. While the authors of the previous work speculated that the diversity of clients could be a reason for the existence of alpha and beta traffic, our results indicate that the cause can probably be found in the characteristics of the servers.

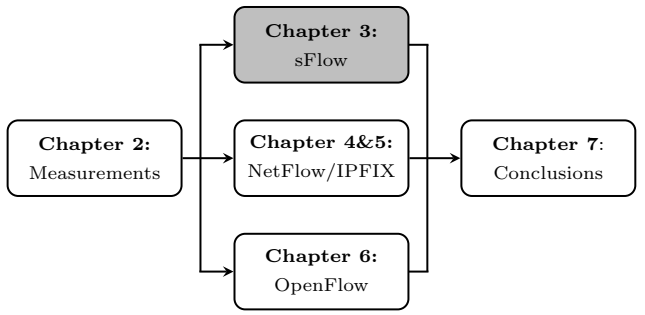
Recognize that the very molecules that make up your body, the atoms that construct the molecules, are traceable to the crucibles that were once the centers of high mass stars that exploded their chemically rich guts into the galaxy, enriching pristine gas clouds with the chemistry of life. So that we are all connected to each other biologically, to the earth chemically and to the rest of the universe atomically. That's kinda cool! That makes me smile and I actually feel quite large at the end of that. It's not that we are better than the universe, we are part of the universe. We are in the universe and the universe is in us.

— Neil deGrasse Tyson

CHAPTER 3

sFlow-based Link Dimensioning

Monitoring technologies that implement packet sampling techniques are widely used by network operators because they are scalable to high-speed links while providing higher granular data than SNMP counters. However, sampled data only gives a partial view of the actual observed traffic. In this chapter we demonstrate the feasibility of using sampled traffic data for link dimensioning. Publications related to this chapter are [42, 43].



The organization of this chapter is as follows:

- *Section 3.1 provides background and literature review on packet sampling, and states the contribution of this chapter.*
- *Section 3.2 introduces the sFlow monitoring tool.*
- *Section 3.3 details the two other sampling algorithms studied in this chapter, namely, Bernoulli and n-in-N sampling.*
- *Section 3.4 proposes approaches to better estimate traffic variance from sampled data.*
- *Section 3.5 shows experimental results validating the use of sampled data and adapted variance formulas for link dimensioning.*
- *Section 3.6 presents a study assessing the impact of the exporting process of sampled data on the link dimensioning.*
- *Section 3.7 concludes this chapter.*

3.1 Background

Packet sampling strategies are adopted by network operators to perform traffic measurements in high-speed links due to the massive volume of transferred data. These strategies help to reduce the excessive load of measurements storage and processing, while still providing highly granular traffic data – we are still talking about packet captures after all. To our goal of an easy-to-use link dimensioning, packet sampling tools are very attractive alternatives to measure traffic.

Packet sampling techniques range from simple approaches, such as systematic or probabilistic, to more complex techniques that select packets based on their content and flow properties. However, in our context of link dimensioning, the most important is the correct choice of the sampling rate. The sampling rate indicates how many packets are to be sampled from the whole observed traffic. For selecting the sampling rate, network operators need to find a common term between the required measurement effort and the granularity of the measured data. That is, by setting higher sampling rate, on the one hand, the measured data is more granular, but this comes at the cost of requiring more measurement efforts since more packets will be sampled. On the other hand, lower sampling rates are computationally less demanding, but they might not provide enough data for having a realistic overview of the observed traffic during the measured period.

There are several software that implement packet sampling for network traffic measurements and sFlow [62, 95, 102] is the most famous among them. sFlow, which is further described in Section 3.2, implements its own sampling algorithm, which behaves similar to the n-in-N sampling [119]. Sampling techniques can also be found as integrating part of other measurement technologies, such as the Random Sampled NetFlow [27] that supports 1-in-N sampling.

The challenge of using packet sampling for link dimensioning is that the formula in Eq. (1.2) (from [109]) requires traffic average rate and traffic variance, which might not be easily computed out of sampled data. Following we present a brief literature review on the use of packet sampling in network operations and management. Then, we clearly state the contribution of this chapter.

3.1.1 Literature Review

Concerning packet sampling techniques, surprisingly, not many previous works have addressed their use on network operations and management. For example, in [21] the authors propose an adaptive sampling approach in which they can estimate traffic load and variance from sampled packets. Such approach is based on the assumption that sampling errors arise from the dynamics of packets sizes and counts, and that sampling with static rate cannot guarantee accuracy on

estimation of traffic statistics. In [22] the authors propose an adaptive sampling algorithm to estimate flow sizes and, more recently, in [114] the authors implement a sampling-based monitoring algorithm that focuses on accurate estimates of throughput for individual flows. Moreover, the use of packet sampling has also been addressed in other areas. For example, in [13] and [77] the authors assess the impact of sampling on anomaly detection operations. To the best of our knowledge, the impact of basic packet sampling techniques on the link dimensioning process has not yet been investigated by previous works.

3.1.2 Contribution

Aiming at an easy-to-use link dimensioning approach, we do not propose novel sampling algorithms. Instead, we focus on sFlow, 1-in-N and Bernoulli sampling algorithms, which are mostly found within traffic measurement tools. The Bernoulli algorithm is not widely adopted as the other two. However, the decision for including this algorithm in our study was motivated by the fact that it is a well-known and standard sampling algorithm [119].

The challenge of using sampled data for link dimensioning is that the formula from Eq. (1.2) requires the traffic rate and variance, both statistics that can be directly affected by the packet sampling. In this chapter we propose approaches that provide more accurate estimates of traffic variance from sampled data taking into account the additional variance introduced by the sampling algorithms.

Besides the study on the impact of packet sampling on link dimensioning, we also investigate the impact of the exporting process of the sFlow tool. The exporting strategy implemented by sFlow introduces additional abstraction of the individual packet timestamp. That is, many sampled packets are sent from the measurement point to a collector as a batch sharing a single timestamp. Ultimately, this indirectly impacts on the accuracy of the link dimensioning, since packet inter-arrival time is lost and the estimation of traffic variance becomes even more challenging.

The impact of the sampling algorithms on link dimensioning, as well as the efficacy of the proposed formulas to estimate variance from sampled data, is assessed using the traffic measurements dataset introduced in Chapter 2.

3.2 sFlow Monitoring Tool

sFlow [102] is a traffic monitoring tool that uses packet filtering and sampling to provide scalable traffic measurements at the packet level for high-speed networks. Scalability is a major requirement for monitoring large amounts of traffic in high-speed links. For example, in [65] the authors presented an overview of

the traffic at AMS-IX that was measured using sFlow. In 2006 the AMS-IX was – and so it remains – the largest Internet exchange in the world. The authors used sFlow to measured traffic aggregates that averaged 136 Gb/s with peaks up to 208 Gb/s. As for today, AMS-IX still uses sFlow to monitor and measure their traffic[1], which averages around 1.8 Tb/s with peaks higher than 2.5 Tb/s. Another example of deployment of sFlow is by the network operators at CERN [59], which need to monitor and measure massive amounts of e-science data every day.

One of the key points that make sFlow so popular is the ease of finding network devices that have sFlow preinstalled as part of their default configuration. An exhaustive list of sFlow-enabled devices can be found in the *sflow.org*, the official sFlow forum[2].

In this section we initially describe the overall sFlow operation and then we detail both the data exporting procedure and the sampling algorithm implemented by commercial sFlow tools. There are many commercial implementations of sFlow, but the *InMon Corporation*'s sFlow [62, 95] and `pmacct` [76] are likely to be the most common and widely deployed ones. Therefore, descriptions and implementations of sFlow in this chapter are in accordance with these two tools. Operations that are not of interest to the context of link dimensioning, such as packet filtering, are not described in details.

3.2.1 sFlow Overall Operation

The general operation of sFlow is shown in Figure 3.1. sFlow operation can be roughly divided between two entities, namely *agent* and *collector*. The sFlow agent is a software located at the router, switch or standalone probe device, and it is responsible for monitoring and measuring the traffic and handling the sampled data. The agent can be remotely accessed and controlled via a sFlow MIB. The sFlow agents export sampled data to a centralized collector using sFlow datagrams. At the collector the sampled data is analyzed and stored.

Every observed packet by the sFlow agent undergoes two stages: filtering and sampling. The former allows the agent to, *e.g.,* drop packets that belong to flows that are not of interest. All packets that are not discarded by the filtering process are subjected to the sampling algorithm. Note that for the context of link dimensioning we are primarily interested in the whole traffic aggregate and, hence, we do not take packet filtering into consideration in our analysis. The sampling algorithm implemented by InMon [62, 95] or `pmacct` [76], as well as by other sFlow tools, is later detailed in Section 3.2.3. Before being exported to the collector, sampled packets are kept in a buffer within the agent. As explained in

[1] https://www.ams-ix.net/technical/statistics/sflow-stats
[2] http://www.sflow.org/products/network.php

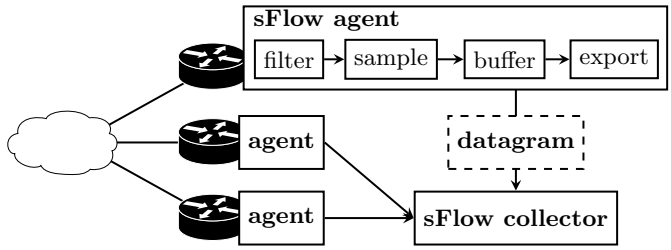


Figure 3.1: sFlow exporting process.

Section 3.2.2, the waiting period in the buffer before being exported results in an additional problem for the link dimensioning approach, which is the missing timestamp of the sampled packets.

3.2.2 sFlow Exporting of Sampled Data

The exporting procedure in sFlow defines when sampled packets are sent from the sFlow agent to the collector. All sampled packets are sent from the agent to the collector using a sFlow datagram with maximum size of 1500 bytes. sFlow defines two conditions for exporting buffered packets at the agents: (1) there are enough packets to fill in a sFlow datagram; or (2) the oldest sampled packet in the buffer should not wait for longer than 1 second to be exported to the collector.

There is one important characteristic from the exporting procedure that might directly impact on the link dimensioning problem. The sampled information of packets does not include their respective timestamps. At the collector, therefore, all the received sampled packets that are grouped into the same datagram share the same timestamp, which is the datagram's timestamp. The amount of missing information depends on the link load, sampling rate and agent's buffer size (the last is defined by the capture length). Clearly, the consequences of not having the packets timestamps is not knowing how the packets are spread in time (*i.e.,* packets inter-arrival time) and, hence, it might become difficult to estimate traffic variance out of the sampled data.

Note that this is the way sFlow is defined in [95] and, hence, implemented by tools that follow such standard. It is perfectly possible to modify the source code of: (1) the sFlow agent to include packets timestamps within sampled packets headers; (2) the sFlow datagram to carry this extra information; and (3) the sFlow collector to process the individual timestamps. However, aiming at reduced efforts on the traffic measurements, and considering that operators

will not modify the software that comes already embedded in their network devices, we consider the implementation of sFlow as defined in [95].

3.2.3 sFlow Sampling Algorithm

The sampling algorithm described in this section is the same implemented by tools such as InMon's sFlow [62] and `pmacct` [76]. In the sFlow documentation the sampling algorithm is referred as *random sampling*. However, to avoid confusion with the 1-in-N random sampling of [119], in this thesis we refer to the algorithm of Figure 3.2 as *sFlow sampling*.

Figure 3.2 shows the pseudo-code of the sFlow sampling algorithm. It is a rather straightforward algorithm in which the decision of sampling a packet is based on a randomly generated counter such that in average 1-in-N packets are sampled. On line 7 the counter s is assigned with a random value from the function on line 2. The counter s tells the algorithm how many packets to skip before sampling one. This is done by progressively decrementing s for every received packet (line 9), until it becomes zero, what means that the current packet must be sampled and a new random counter assigned to s (lines 10–12). Typically, the random number generator yields uniformly distributed numbers and is seeded with the system's current time (line 6).

```
    in: maximum skip N
    in: input stream of packets seen by the monitoring point
    out: output stream of sampled packets

 1: function GetSkip()
 2:     return randomNatural(1..2 · N − 1)
 3: end function
 4:
 5: procedure sFlowSampling(input,output)
 6:     seed random number generator
 7:     s := getSkip()
 8:     while packet pkt available from input do
 9:         s := s − 1
10:         if s = 0 then
11:             sample packet pkt to output
12:             s := GetSkip()
13:         end if
14:     end while
15: end procedure
```

Figure 3.2: sFlow sampling algorithm.

3.3 Alternative Sampling Methods

In this section we introduce two additional sampling algorithms further used in this chapter for assessing the impact of sampled data on link dimensioning. Although there are many works proposing new sampling strategies that take into account packets and flows properties, given our goal of an easy-to-use link dimensioning, we only consider algorithms that can be found in traffic monitoring tools and that have been standard. That is the case for the above presented sFlow sampling, as well as it is the case for the Bernoulli and the n-in-N sampling algorithms [119] both described in this section.

3.3.1 Bernoulli Sampling

In Bernoulli sampling, described in [119], each packet is independently sampled with constant probability p. Figure 3.3 shows the pseudo-code for Bernoulli sampling as implemented by us in this thesis. For every received packet, on line 8 the variable s is assigned with a real random number between 0 and 1, obtained from the function in line 2. The current packet is sampled if the value of s is less than the defined sampling probability p (lines 9–10).

```
    in: probability p
    in: input stream of packets seen by the monitoring point
    out: output stream of sampled packets

 1: function GetNumber()
 2:     return randomReal(0..1)
 3: end function
 4:
 5: procedure BernoulliSampling(input,output)
 6:     seed random number generator
 7:     while packet pkt available from input do
 8:         s := GetNumber()
 9:         if s < p then
10:             sample packet pkt to output
11:         end if
12:     end while
13: end procedure
```

Figure 3.3: Bernoulli sampling algorithm.

With Bernoulli sampling algorithm consecutive or nearby packets can be sampled more frequently than with, *e.g.,* sFlow sampling. This is because the

former does not have a counter or variable that sets a limit between two samples (*i.e.,* a kind of sampling range). Moreover, specially in the context of link dimensioning where traffic fluctuations are very important, sampled data with Bernoulli might become misleading since there is a chance that none of the packets constituting a traffic burst are sampled. This problem is later studied in the experiments of Section 3.5.

3.3.2 n-in-N Sampling

In n-in-N sampling, also described in [119], the stream of observed packets is divided in non-overlapping windows of N packets, and from each window n packets are randomly sampled. This random selection helps to avoid sampling packets that belong to periodic traffic patterns, which is the drawback of the systematic sampling. Despite being more complex than Bernoulli sampling, n-in-N sampling is widely used by operators and in traffic measuring tools, such as NetFlow, because its design prevents more than $2n$ consecutive packets to be selected, ultimately avoiding measurement bursts. The sampling range defined by N also guarantees that packets are sampled during traffic bursts, given that these bursts are larger than N packets.

Figure 3.4 the pseudocode as implemented by us in this thesis. The range of indices of size n packets to sample, within the window of size N packets, is randomly generated (line 3). Although not specified in the pseudocode of Figure 3.4, the resulting array of n values in line 3 consists of unique values within the interval $[1..N]$. The incremental variable c defines an index for every observed packet (line 13). A packet is sampled if its index matches one in the defined range (lines 14–15). A new range of indices is defined once the counter c has reached the same value of the sampling window size N (lines 17–19) and the whole procedure repeats.

Note that in the experiments presented in Section 3.5 we only consider the case of $n = 1$ since this is the most commonly used one. For example, Random Sampled Netflow only supports 1-in-N sampling [27].

3.4 Estimating Traffic Variance

Traffic mean rate and traffic variance are the two parameters needed by the dimensioning formula used in our work (formula introduced in Section 1.4). From the original proposal of this formula, in [109], traffic mean ρ and the traffic variance $v(T)$ at the chosen timescale T are easily obtained from complete traffic captures, *i.e.,* uninterrupted packet capturing. In order to apply the link dimensioning formula to sampled data, however, traffic mean and variance need to be estimated.

```
    in: window size N
    in: number of samples n
    in: input stream of packets seen by the monitoring point
    out: output stream of sampled packets

 1: function GetNumbers()
 2:     for i = 1 → n do
 3:         a[i] := randomNatural(1..N) // unique values
 4:     end for
 5:     return a
 6: end function
 7:
 8: procedure ninNSampling(input,output)
 9:     seed random number generator
10:     s := getNumbers()
11:     c := 0
12:     while packet pkt available from input do
13:         c := c + 1
14:         if c in array s then
15:             sample packet pkt to output
16:         end if
17:         if c = N then
18:             s := getNumbers()
19:             c := 0
20:         end if
21:     end while
22: end procedure
```

Figure 3.4: n-in-N sampling algorithm.

Considering sampled data now, let r be the ratio between the total number of monitored packets and the number of sampled packets (*i.e.*, the inverse of the sampling rate). Let ρ' be the mean traffic rate of the sampled traffic and let $A'_i(T)$ be the amount of sampled traffic (in bytes) observed in time interval i of length T. The original amount of traffic $A_i(T)$ in that interval can be estimated by

$$A_{i,est}(T) = r \cdot A'_i(T) \; . \tag{3.1}$$

Hence, the original mean traffic can be estimated by

$$\rho_{est} = \frac{r}{nT} \sum_{i=1}^{n} A'_i(T) \; . \tag{3.2}$$

Similarly, a (naive) estimation of the original variance can be obtained by

$$v_{est}(T) = \frac{r^2}{n-1} \sum_{i=1}^{n} (A'_i(T) - \rho' T)^2 \ . \tag{3.3}$$

The drawback of such simplistic approach is that, while ρ_{est} is an unbiased estimator of the mean traffic rate ρ, the traffic variance may be overestimated by $v_{est}(T)$ especially for small T and large r. That is because the additional variance introduced by the sampling process is not taken into account. In the following we propose better estimators for the traffic variance according to the sampling algorithm used.

3.4.1 Variance Estimation with Bernoulli Sampling

Without sampling, the number of bytes A_i in interval i is

$$A_i = \sum_{j=1}^{P_i} S_{i,j} \ , \tag{3.4}$$

where P_i is the number of packets in interval i and $S_{i,j}$ is the size of the jth packet in the interval. If we assume that P_i are i.i.d. (independent and identically distributed) like a random variable P, $S_{i,j}$ are i.i.d. like a random variable S, and P and S are independent, well-known results for random sums can be applied and it holds

$$\rho = \frac{1}{T} E[P] E[S] \ , \tag{3.5}$$

$$v(T) = E[P] Var[S] + E[S]^2 Var[P] \ . \tag{3.6}$$

These two equations can also be used in the case of Bernoulli sampling. We "simulate" Bernoulli sampling with sampling probability p by randomly setting the size of some packets to zero. The size of a sampled packet becomes

$$S'_{i,j} = \begin{cases} 0, & \text{with probability } 1-p \\ S_{i,j}, & \text{with probability } p \ . \end{cases}$$

Replacing S by S' in Equation (3.6) (the complete derivation can be found in the appendix) yields the variance estimation

$$v_{bern}(T) = v_{est}(T) - (r-1) E[P] E[S^2] \ , \tag{3.7}$$

where $r = 1/p$, $v_{est}(T)$ is the naive estimation from Equation (3.3), $E[P]$ is the average number of packets per time interval before sampling, and $E[S^2]$ is

the second moment of the packet size before sampling. $E[P]$ can be estimated by multiplying the (measured) average number of sampled packets per time interval by r. $E[S^2]$ can be estimated directly from the sizes of the sampled packets since the sampled packets have the same size distribution as the non-sampled packets according to our assumptions. Alternatively, traffic models could be used to obtain $E[S^2]$. For example, assuming a packet size uniformly distributed between 40 and 1500 bytes, we would have

$$E[S^2] = \frac{1}{12}(1500 - 40)^2 + \frac{1}{4}(1500 - 40)^2 \ . \tag{3.8}$$

3.4.2 Variance Estimation with 1-in-N and sFlow Sampling

A mathematical treatment of 1-in-N sampling is much more complex than for Bernoulli sampling. The sampling window of N packets can stretch over several intervals T, making the sampled traffic $A'_i(T)$ and $A'_{i+1}(T)$ in adjacent time intervals dependent. Similar difficulties arise for sFlow sampling. Hence, we simplify the problem and assume that $1/N$ of the packets of each interval are sampled. Furthermore, we assume again that the numbers of packets per interval in the original traffic stream are i.i.d. like P and packet sizes are i.i.d. like S, and P and S are independent. Under these assumption, the number of sampled packets P'_i in time interval i is $P'_i = P_i/r$ with $r = N$ (we ignore the problem that P_i/r might not be a natural number). Replacing P by P/r in Eq. (3.6), we obtain the variance estimation

$$v_N(T) = v_{est}(T) - (r-1)E[P]Var[S] \ , \tag{3.9}$$

where $E[P]$ is the average number of packets per time interval before sampling, and $Var[S]$ is the variance of the packet size before sampling. Again, $E[P]$ can be estimated from the number of sampled packets and $E[S^2]$ can be estimated, and hence $Var[S]$, directly from the sizes of the sampled packets or, alternatively, from traffic models. For example, assuming that packet sizes are uniformly distributed between 40 and 1500 bytes, we would obtain

$$Var[S] = \frac{1}{12}(1500 - 40)^2 \ . \tag{3.10}$$

3.5 Experimental Results

In this section we assess the use of the sampled data for link dimensioning purposes. We also validate our proposed formulas for estimating traffic variance out of sampled packets.

The content of this section is organized as follows. In Section 3.5.1 we set out the approach we use to validate the results obtained for estimations of required capacity using the dimensioning formula from Section 1.4 and sampled packets obtained from the different sampling algorithms earlier presented in this chapter. The metrics introduced in this section are later used to validate results from other proposed procedures in Chapters 4 and 5. In Section 3.5.2 we demonstrate the impact of sampling on Gaussianity of traffic, which is an important requirement of the adopted dimensioning formula. In Section 3.5.3 we compare and discuss results on the estimation of required capacity for each one of the considered sampling strategies and, in Section 3.5.4 we present the results of an extensive validation of using sampled data for link dimensioning. The validation in this section comprises all traces of our dataset.

3.5.1 Approach

In our experiments, we sampled all traces from our dataset (introduced in Section 2.3. We used the three sampling algorithms previously described in this chapter, namely, sFlow, Bernoulli and n-in-N sampling. The sampling rates we applied were 1:10, 1:100 and 1:1000. The last one, however is not applicable to many traces of our dataset since the total traffic of some locations comprises only a few thousands of packets (see Section 2.3.2). The resulting problems with low sampling rates are shown in the next sections.

In a first moment, we assess the Gaussianity fit of sampled traffic. This is important due to the assumptions on which the link dimensioning formula in Equation (1.2) from [109] builds upon. That is, if after sampling the Gaussian character of traffic is diminished, we should expect a negative impact on the accuracy of the estimations of required capacity. In a second moment, we quantify the accuracy of the link dimensioning using sampled data. Since excessive overestimation of required capacity is as undesired as underestimation, in a third moment, we also quantify the overshooting of estimations from sampled data. The approaches for addressing each one of these three aspects are given next.

Assessing Gaussianity of Sampled Traffic

To assess the Gaussian fit of traffic traces we use the exact same procedure as described in Section 2.4.2. For each sampled trace from our dataset we calculate its respective Gaussianity goodness of fit γ (*linear correlation coefficient* from Equation (2.2)).

Quantifying Underestimation of Required Capacity

To validate whether the link dimensioning procedure using sampled data is successful, we compare the resulted estimation of required capacity $C(T, \varepsilon)$ with an empirical one computed from the complete traces (*i.e.,* not sampled traces). The empirical estimation of required capacity is the 99th-percentile of the empirical CDF distribution of the throughput. This value represents the minimum capacity that should be allocated so that in only a predefined amount of time intervals of size T (*i.e.,* represented by ε set in the link dimensioning formula) the throughput will be above the required capacity $C(T, \varepsilon)$. The empirical estimation is defined as

$$C_{emp}(T, \varepsilon) := \min \{C : \#\{A_i \mid A_i > CT\}/n \leq \varepsilon\} \ , \tag{3.11}$$

where $A_1, \ldots, A_n$ are the empirical traffic aggregates on timescale T and ε is the bandwidth exceedance probability set in the dimensioning formula. Additionally, to verify the accuracy of an estimated capacity C, we calculate the amount of measured intervals in which the traffic aggregate A_i exceeds C by

$$\hat{\varepsilon} := \#\{A_i \mid A_i > CT\}/n \ . \tag{3.12}$$

Clearly, if $\hat{\varepsilon} \leq \varepsilon$ the procedure did not underestimate the required capacity. Note that $\hat{\varepsilon} \leq \varepsilon$ is, by definition, equivalent to $C(T, \varepsilon) \geq C_{emp}(T, \varepsilon)$. In order to comply with previous work of [96, 109, 112], in the following experiments we set $\varepsilon = 0.01$ (*i.e.,* 1%) and T ranging from 1ms to 1s.

Quantifying Overestimation of Required Capacity

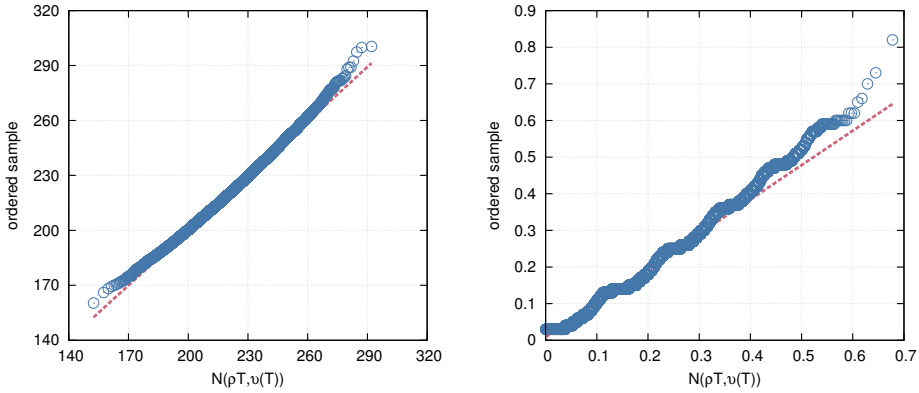
Excessive overestimation of the required capacity is as much undesired as its underestimation. That's because overestimation may result in, for example, waste of link resources that could potentially be reallocated to other purposes. Therefore, we are also interested in quantifying the overshooting of the estimation of required capacity $C(T, \varepsilon)$. To do so, we calculate the relative error, in percentage, for any T and ε, between the estimation of required capacity and the empirical value for the same trace, which is given by

$$RE = \frac{C - C_{emp}}{C_{emp}} \cdot 100\% \ , \tag{3.13}$$

where C is the estimated required capacity obtained using the dimensioning formula applied to either complete or sampled traces.

3.5.2 Traffic Gaussianity

The link dimensioning approach in [109], from which we borrow the dimensioning formula of Equation (1.2), builds upon the assumption that traffic is Gaussian distributed. That's why it is important to first check whether sampled traffic remains Gaussian before estimating the required capacity. Since only a fraction of the real traffic remains after sampling, Gaussian character of traffic might be distorted. That is, after sampling what was before good Gaussian traffic might become bad, and vice versa. As a consequence of bad Gaussianity, the resulting estimations of required capacity become unpredictable. Not only underestimation might be a result of bad Gaussianity, but also excessive overestimation. For the Gaussianity assessment in this section, we follow the same approach as described in Section 2.4.2.



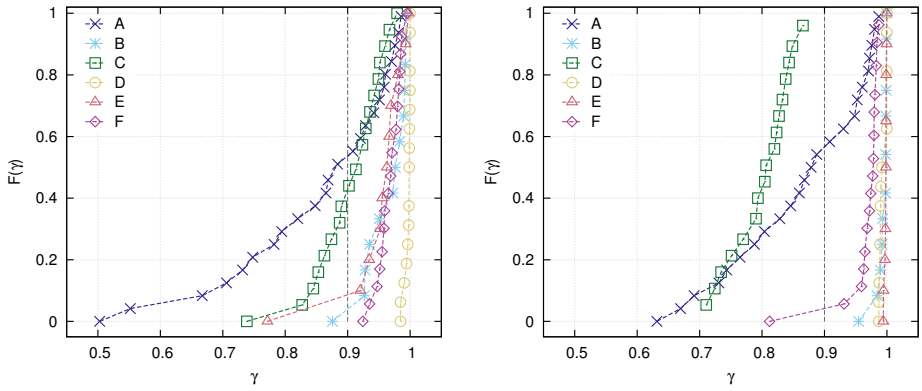
(a) Not sampled

(b) Sampled 1:1000

Figure 3.5: Q-Q plots at $T = 100$ms for an example trace before and after sampled 1:1000.

For matters of illustration, in Figure 3.5 we first show the Q-Q plot at $T = 100$ms for an arbitrarily selected trace from our dataset before and after sampled 1:1000. As one can see in Figure 3.5a, traffic of the example trace is likely to have a good Gaussian fit, since only few points fall out of the diagonal line. However, when sampled 1:1000 using 1-in-N sampling, we can observed many deviations from the diagonal line and, therefore, Gaussianity characteristic of the traffic seems to be lost. This is an extreme example where we show that a very low sampling rate may have major impact on the traffic characteristics and, consequently, may lead to inaccurate estimations of required capacity. It

is also important to know that the impact of sampling on traffic Gaussianity fit is aggravated at shorter timescales.



(a) Not sampled

(b) Sampled 1:1000

Figure 3.6: CDF of γ at $T = 100$ms for all traces per location.

The main take away of the Gaussianity assessment in this section is that when using a proper sampling rate (*i.e.,* sampling rate that takes into consideration the volume of traffic in the monitored link), Gaussianity characteristics of traffic tend to remain unchanged or be meaninglessly affected. In fact, at $T = 100$ms around 70% of traces in our dataset have good Gaussian fit with an overall average γ of 0.9135, and when these are sampled 1:10 using 1-in-N algorithm, the average γ becomes 0.9164 and around 71% of all traces have $\gamma \geq 0.9$.

To quantify the loss of such characteristic for all traces in our dataset after sampled, we calculate the Gaussianity goodness of fit γ (see Equation (2.2)) for each trace individually. Figure 3.6 shows the CDF of γ for all traces in our dataset before and after sampled 1:1000 using 1-in-N sampling. Traces from locations A and C, that have smaller traffic aggregates, have their Gaussianity fit severely impacted by using such a low sampling rate. Moreover, by comparing the plots of Figures 3.6a and 3.6b one can see that the opposite situation also happens: Gaussianity fit of traces actually increased after sampled. That is, when an inappropriate sampling rate is chosen crucial traffic characteristics might be lost, ultimately leading to a misleading Gaussianity fit.

3.5.3 Comparing Sampling Strategies

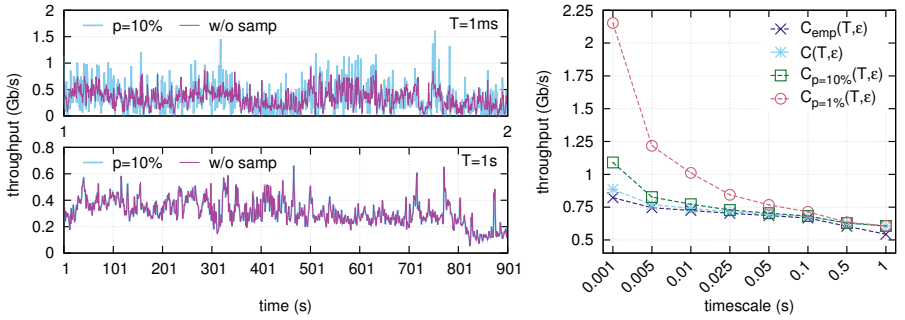
In this section we show the differences on the obtained sampled data from each of the three implemented sampling algorithms. Figure 3.7 shows, for a single example trace, the traffic time series (left) and estimations of required capacity (right) at various T for each sampling algorithm. From analyzing the time series one can see that the problem of overestimation is likely to happen at shorter T. Independently of sampling algorithm, at $T = 1$ms we can see several bogus traffic peaks in the time series from sampled data (*i.e.,* inexistent bursts in the actual observed traffic). That is, traffic peaks that are not observed in the time series from traces without sampling. Note that, for a better visualization, the time series at $T = 1$ms in Figure 3.7 only plot one second.

The overestimation of traffic rates within individual time intervals is more problematic with Bernoulli sampling. That is because with Bernoulli sampling several consecutive packets, or packets very close in time, might be sampled. Such packets might end up in the same time interval and, ultimately, the scaling of traffic within the interval, *i.e.,* by multiplying the sampled traffic by r (see Equation (3.2)) creates bogus traffic bursts.

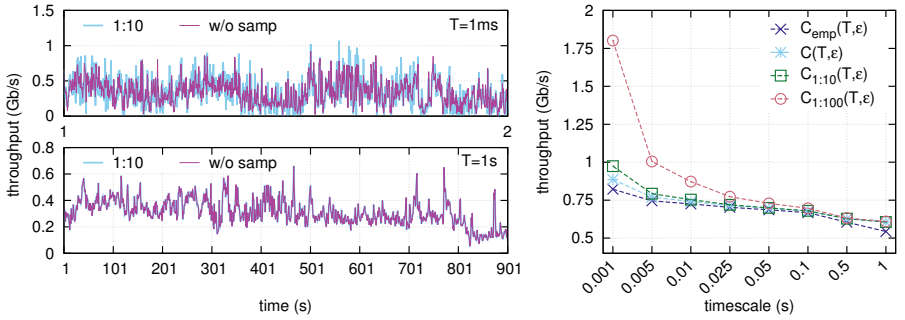
This problem is alleviated with 1-in-N sampling due to the definition of sampling windows of N packets. Although with lower probability, overestimation of traffic average can also happen for 1-in-N sampling once nearby packets are sampled, *e.g.,* when the last packet of a current window and the first packet of the next window happen to be sampled. Once again, these sampled packets might end up in the same time interval, which later creates fake traffic bursts.

Using sFlow sampling results are in between the ones from Bernoulli and 1-in-N sampling strategies. Although also defining a kind of sampling window, sFlow does not discard the remaining packets in the current window, as done by the 1-in-N algorithm. That is, if the sampled packet is not the last in the window, all the remaining packets are considered as part of the packets range for the next sampling. This makes the behavior of sFlow sampling to be somewhere in between Bernoulli and 1-in-N sampling.

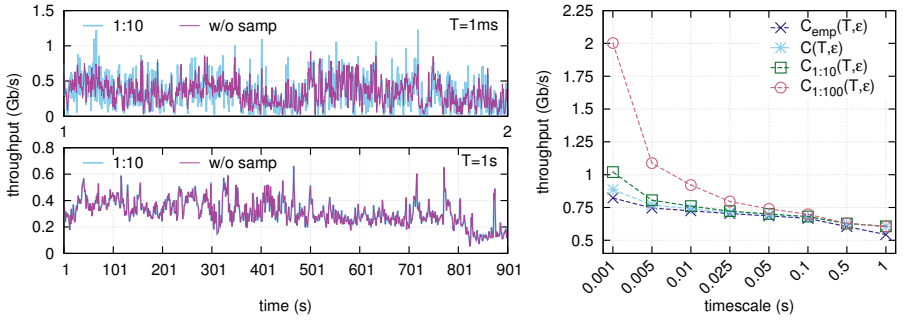
The problem of bogus traffic bursts disappears at larger timescales. As one can see in Figure 3.7, at $T = 1$s time series created from sampled data (independently from the concrete sampling technique), using an arbitrarily chosen trace, faithfully reproduce traffic fluctuations as seen for the non-sampled traces. In Figure 3.7, time series from sampled data are estimated using Equation (3.1). These results become clearer when analyzing the estimations of required capacity $C(T, \varepsilon)$ for various T and sampling rates (right plots of Figure 3.7). It is clear that, for the example trace used in these figures, which is the same for the three algorithms, sampling rates of 1:100 are too low and, hence, they result in excessive overestimation of the required capacity at shorter timescales.



(a) Bernoulli sampling



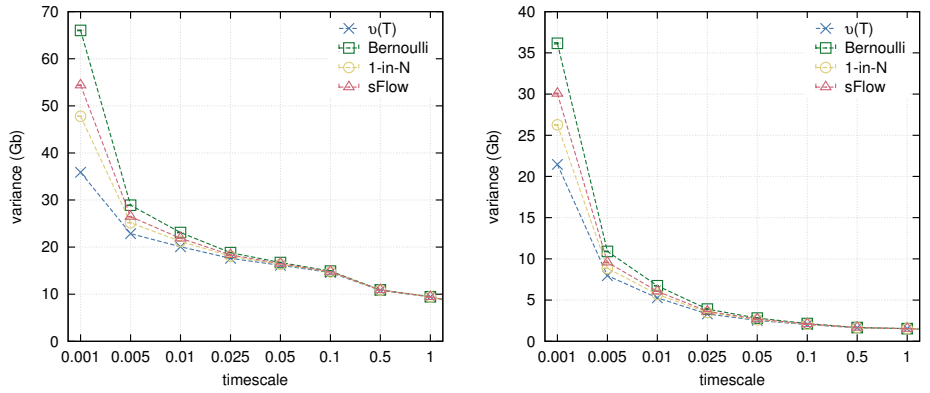
(b) 1-in-N sampling



(c) sFlow sampling

Figure 3.7: Traffic time series (left) and estimations of required capacity (right) for all sampling algorithms and using an example trace from A. For the sake of clarity, at $T = 1$ms only one second is shown in the time series.

Nonetheless, the difference of final results of $C(T, \varepsilon)$ between the three sampling techniques is not significant. Note that $C(T, \varepsilon)$ in these plots is for estimations using the complete trace and when using the sampled data the legend specifies the sampling rate.



(a) Trace from A (b) Trace from F

Figure 3.8: Difference between traffic variance calculated from 10 runs of sampling for each algorithm with trace sampled 1:10.

This small difference between the results obtained for each sampling approach remains constant for other traces and for several runs of sampling. Figure 3.8 shows the average traffic variance calculated from 10 runs of sampling for each algorithm always using the same example trace. For matters of comparison, the actual traffic variance $v(T)$ of the example traces is also plotted. Although very small and difficult to see, the standard deviation is plotted as error bars in the plots of this figure. For both example traces, at any T or sampling algorithm, the standard deviation is smaller than 1% of the average variance. This confirms that, even with the random nature of the sampling algorithms, running them many times yields very close results. Concerning the traffic variance, the difference between the three sampling algorithms is larger at $T = 1$ms, although still not very significant. At any other T this difference is negligible or inexistent. These differences are also in line with the estimations $C(T, \varepsilon)$ presented in the right plots of Figure 3.7.

Given the small difference between results from different sampling algorithms, for simplicity in the next section, the overall assessment of the impact of packet sampling on link dimensioning is done using only 1-in-N sampling. In addition, due to the very small difference between variances obtained from

10 runs of sampling, results presented in the next section are obtained from a single run of sampling per trace in our dataset.

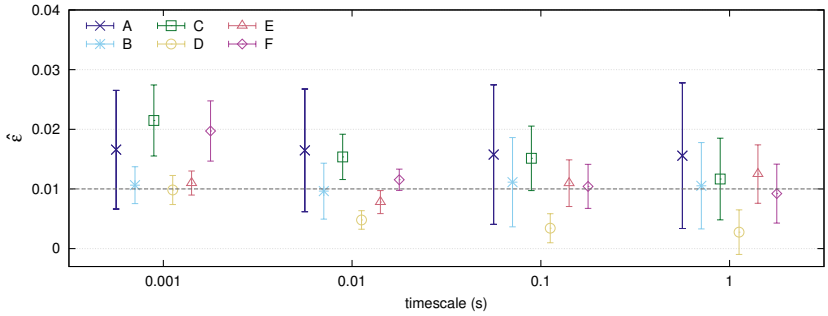
3.5.4 Overall Results

In this section we present results of an extensive validation of the use of sampled data for link dimensioning. To do so, we sampled all traces in our measurements dataset using 1-in-N.

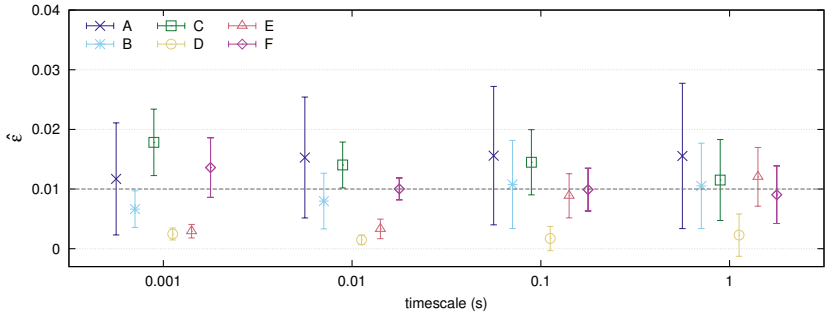
Even using the adapted variance formula from Equation (3.9), conservative $C(T, \varepsilon)$ due to overshooting of the traffic variance might be a problem at smaller timescales and lower sampling rates. Figure 3.9 clearly shows this problem. This figure shows the average and standard deviation (error bars) of $\hat{\varepsilon}$ at various T and sampling rates. It is important to recapitulate that $\hat{\varepsilon}$, as defined in Equation (3.12), is the fraction of measured intervals of size T in which actual traffic rates are higher than the estimated $C(T, \varepsilon)$. By setting $\varepsilon = 1\%$ in the dimensioning formula of Equation (1.2), a successful estimation of required capacity ultimately yields $\hat{\varepsilon} \leq \varepsilon$, which is equivalent to $C(T, \varepsilon) \geq C_{emp}(T, \varepsilon)$.

As shown in Figure 3.9, very conservative results are obtained at small timescales and low sampling rates and they might confront with the intelligent overestimation premiss. For example, Figure 3.9c show that at $T = 1$ms the average and the standard deviation of $\hat{\varepsilon}$ for all locations is near to or equals zero. That is, $C_{emp}(T, \varepsilon)$ was never underestimated. In such cases one should suspect that overestimation might have been too high (as further demonstrated in section 3.5.5). Having in mind that excessive overestimation might happen, from the results in Figure 3.9 one can still argue that estimations from sampled data are close to the ones obtained from complete traces (*i.e.,* without sampling). Therefore, from these results we can conclude that packet sampling can be used for link dimensioning purposes, provided that the sampling rate is wisely chosen, taking into consideration the volume of traffic in the link.

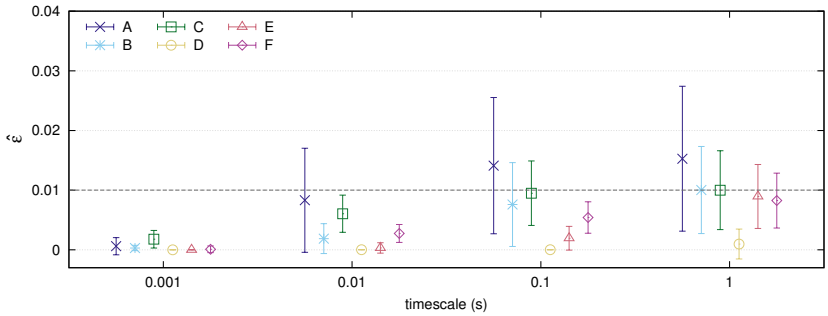
For matters of comparison, Figure 3.10 shows the average and the standard deviation of $\hat{\varepsilon}$ for all traces when estimating the required capacity $C(T, \varepsilon)$ using the traffic variance $v_{est}(T)$ from Equation (3.3), obtained by simply scaling the variance from sampled data by r^2. Given that the additional variance introduced by the sampling process is not taken into consideration, the naive estimation of $v_{est}(T)$ results in excessive overestimation of the required capacity, independently of sampling rate or timescale. Once again, overestimation does not necessarily mean success, but rather the obtained $C(T, \varepsilon)$ might be way higher than the actual needed $C_{emp}(T, \varepsilon)$ for the given traffic aggregate.



(a) Without sampling

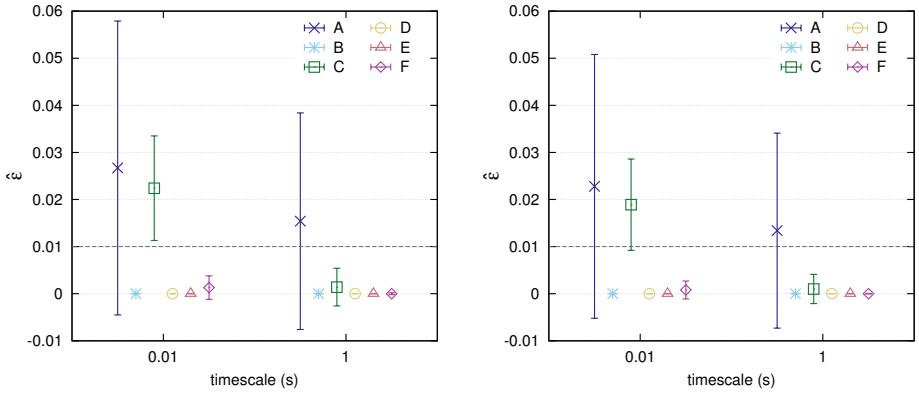


(b) Sampling 1:10 using 1-in-N algorithm



(c) Sampling 1:100 using 1-in-N algorithm

Figure 3.9: Average and standard deviation (error bars) of $\hat{\varepsilon}$ at various T.



(a) Sampled 1:10

(b) Sampled 1:100

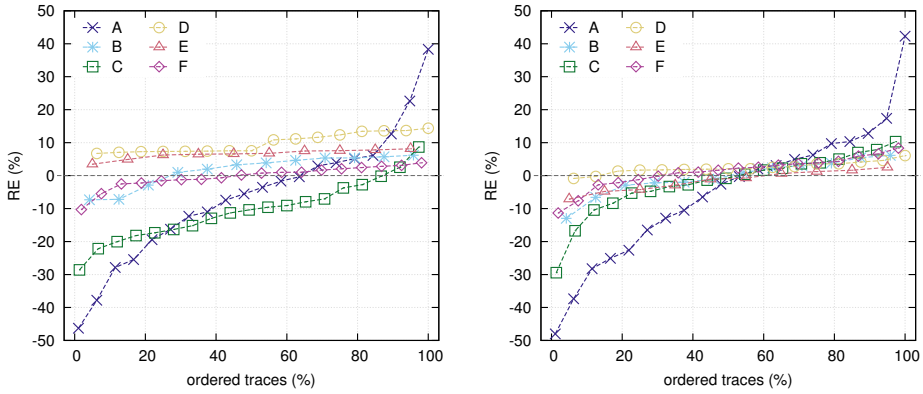
Figure 3.10: Average and standard deviation (error bars) of $\hat{\varepsilon}$ at various T when using $v_{est}(T)$ from Equation (3.3) to estimate traffic variance from sampled data.

3.5.5 Quantifying the Overestimation

Aiming at an intelligent link capacity provisioning, it is important to also quantify how much the estimated required capacity $C(T, \varepsilon)$ overestimates the actual required capacity $C_{emp}(T, \varepsilon)$. The relative error RE, given by Equation (3.13), quantifies the difference between the two mentioned estimations. Figure 3.11 shows the obtained RE for all traces sampled 1:10. One can see that, at $T = 10$ms, around 78% of all traces had under or overestimation within reasonable bounds, *i.e.,* $RE = \pm 15\%$, and at $T = 1$s this value is around 85%. Approximately 65% and 74% of all traces are within a boundary of $RE = \pm 10\%$ at $T = 10$ms and $T = 1$s, respectively. The worst cases are few traces from A for which RE was up to $\pm 50\%$.

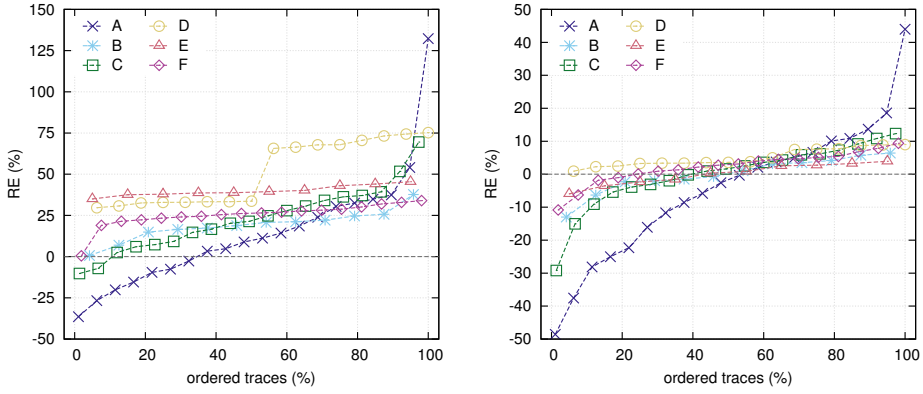
Figure 3.12 shows the relative error RE for all traces sampled 1:100. While RE at $T = 1$s does not differs much from when traces are sampled 1:10, the RE obtained at $T = 10$ms demonstrates the problems arising from a combination of short T and low sampling rates. While for sampling 1:10 at $T = 10$ms most traces had $RE = \pm 15\%$, for traces sampled 1:100 this range increases to $-10\% \leq RE \leq 50\%$. The worst cases are some traces from location D and again few traces from location A for which RE was up to 130%.

One important take away is that the combination of short timescale with inappropriate sampling rate seems to not result in underestimation. These results, specially those in Figure 3.12a, support the statements regarding the problem of excessive overestimation made in the previous section and showed in Figure 3.9.



(a) $T = 10$ms (b) $T = 1$s

Figure 3.11: *RE* for all traces per location. Traces sampled 1:10 using 1-in-N sampling.



(a) $T = 10$ms (b) $T = 1$s

Figure 3.12: *RE* for all traces per location. Traces sampled 1:100 using 1-in-N sampling. Note the different scale of y-axis for figure (a).

3.6 Impact of the sFlow Exporting Process on Link Dimensioning

In the previous section we presented the results on how sampled data can impact the accuracy of the link dimensioning procedure. Considering a real networking scenario in which the traffic is monitored using the sFlow tool, introduced in the beginning of this chapter, the exporting process may also add some challenges to the link dimensioning problem. That is because many sampled packets can be exported in a single sFlow datagram and, hence, their individual timestamps are lost. We have implemented a discrete simulator of sFlow, containing only the functionalities needed for this study, and in this section we assess the impact of the sFlow exporting process (see Section 3.2.2) on link dimensioning using the datasets of location B and C.

3.6.1 Implementation Assumptions

For the specific case of link dimensioning in this thesis, we are interested in the whole traffic and, therefore, all observed packets undergo sampling. That is, our packet filtering rule is defined as "accept all". Aiming at simplicity we have implemented a sFlow exporter focusing on evaluating the effects of the loss of packets timestamps on link dimensioning. Therefore, we make a few assumptions and abstractions: (i) we assume the data analysis at the collector to solely consist of the link dimensioning procedure described and assessed in the previous sections; (ii) sFlow agent and collector are located in the same machine and, therefore, sFlow datagrams do not need to experience network transmission delays; and (iii) sFlow datagrams are simplified to only carry the headers of sampled packets. Note that, considering the characteristics we are interested in, none of above mentioned decisions and assumptions invalidates our experimental implementation of sFlow when compared to a commercial one. It is also important to mention that we do not consider datagram loss due to, for example, packet drops that may happen in congested networks or because the agent's jobs run with a low priority on the exporting router or switch. Furthermore, all metrics used in the following analysis were already introduced in Section 3.5.1.

3.6.2 Exporting Conditions

In the following experiments we consider two conditions for exporting sampled data from the sFlow agent to the collector. In the first condition E_1, packets are exported individually and as soon as they are sampled. This case aims at simulating the optimal scenario for link dimensioning, given that all sampled

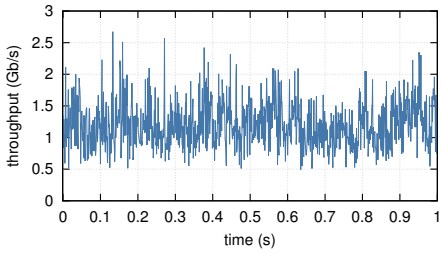
packets will have an individual and chronologically ordered timestamp. Clearly, for a real deployment this condition has the disadvantage of excessive traffic between agent and collector, specially in a distributed monitoring environment. Note that E_1 is the same condition adopted in the experiments of the previous section, where the goal was to validate the sampling algorithms only. The second condition E_2 implements the original sFlow operation as defined in [95], and described in Section 3.2.2. In E_2 the sampled packets are exported when the maximum buffer size is reached (*i.e.,* a full sFlow datagram), or when the oldest sampled packet in the buffer reaches the maximum waiting time of 1 second. Considering a capture size of 128 bytes, and a maximum datagram size of 1500 bytes, the agent's buffer is full when 10 packets are sampled. Additionally, in the following experiments we vary the capture size.

3.6.3 Experimental Results

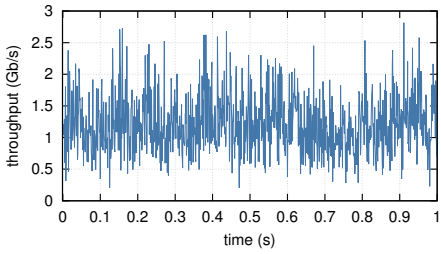
Figure 3.13 shows several traffic time series an arbitrarily chosen trace from location B, created from data monitored under various circumstances. Note that although our traces have a duration of 15 minutes, since plots of Figure 3.13 are for $T = 1$ms, for visualization reasons only one second of the time series is plotted (the same second for all plots though). From these plots one can clearly see the challenges brought to link dimensioning from the sampling and exporting processes. Figure 3.13a shows the time series of the complete data (*i.e.,* not sampled trace). The time series of the sampled data exported using the condition E_1 are shown in Figures 3.13b and 3.13c. Figures 3.13d–3.13f show the traffic time series of sampled data exported using the condition E_2 with the maximum buffer size set to 10 packets ($BS = 10$).

By comparing the time series of the complete traffic with the ones sampled 1:10 (*i.e.,* Figures 3.13b and 3.13d) one can see that the latter two show a more bursty behavior. That is, inexistent traffic fluctuations are created possibly due to clustering of sampled packets into single time intervals (*i.e.,* sampled packets are too close in time). Visually, at this sampling rate, there is not much difference between the two exporting conditions and both satisfactorily approximate the ground-truth observed in the time series of the complete trace. Nonetheless, time series created with exporting condition E_2 are slightly more bursty than with E_1.

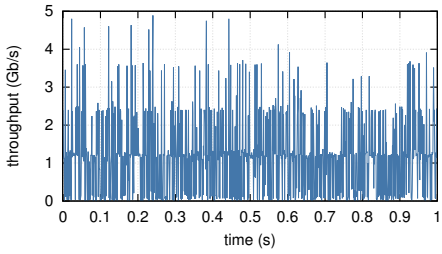
The problem of sampling traffic at unreasonable rates, combined with the problem of loss of individual packet timestamps, becomes evident in the time series of traces sampled 1:100 and 1:1000. Clearly, these sampling rates are too low for the volume of traffic in location B. In Figure 3.13c the ON/OFF behavior is very strong, and it becomes even worse in Figure 3.13e. In this case, many time intervals have a throughput of zero and others have their through-



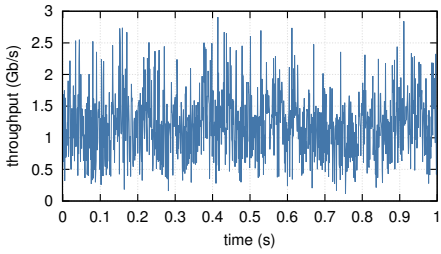
(a) without sampling



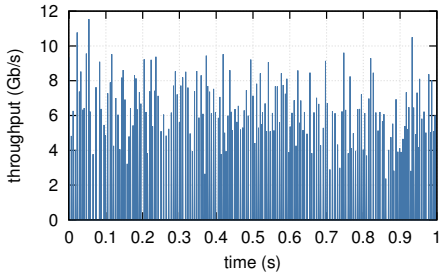
(b) E_1 sampled 1:10



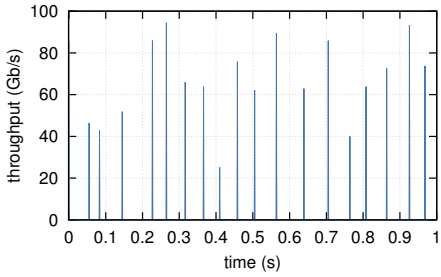
(c) E_1 sampled 1:100



(d) E_2 sampled 1:10



(e) E_2 sampled 1:100



(f) E_2 sampled 1:1000

Figure 3.13: Time series at $T = 1$ms for a trace from location B. For the sake of clarity, only one second is shown.

put overestimated from the scaling process of ρ_{est} in Equation (3.2). That's because at least 10 packets, that originally were some milliseconds apart, are grouped into the same time interval defined by the datagram's timestamp, while neighboring time bins are empty, *i.e.,* no datagrams were exported during those empty time bins. Figure 3.13f shows the time series of the traffic sampled 1:1000 and exported using condition E_2. In this case, the same problem is even further

Table 3.1: Average $\hat{\varepsilon}$ for traces from B.

r	BS	$T = 1$ms		$T = 10$ms		$T = 1$s	
		avg $\hat{\varepsilon}$	$\sigma(\hat{\varepsilon})$	avg $\hat{\varepsilon}$	$\sigma(\hat{\varepsilon})$	avg $\hat{\varepsilon}$	$\sigma(\hat{\varepsilon})$
1:10	2	0.0052	0.0028	0.0072	0.0046	0.0105	0.0072
	5	0.0052	0.0028	0.0072	0.0046	0.0105	0.0072
	10	0.0024	0.0017	0.0070	0.0045	0.0105	0.0072
	20	0.0051	0.0028	0.0072	0.0046	0.0105	0.0072
1:100	2	0.0001	0.0002	0.0010	0.0016	0.0098	0.0073
	5	0.0001	0.0002	0.0010	0.0015	0.0098	0.0073
	10	0.0000	0.0000	0.0001	0.0002	0.0097	0.0073
	20	0.0001	0.0002	0.0010	0.0016	0.0099	0.0075

Table 3.2: Average $\hat{\varepsilon}$ for traces from C.

r	BS	$T = 1$ms		$T = 10$ms		$T = 1$s	
		avg $\hat{\varepsilon}$	$\sigma(\hat{\varepsilon})$	avg $\hat{\varepsilon}$	$\sigma(\hat{\varepsilon})$	avg $\hat{\varepsilon}$	$\sigma(\hat{\varepsilon})$
1:10	2	0.0162	0.0052	0.0133	0.0038	0.0114	0.0069
	5	0.0162	0.0052	0.0133	0.0038	0.0115	0.0069
	10	0.0162	0.0052	0.0133	0.0038	0.0114	0.0069
	20	0.0162	0.0052	0.0133	0.0038	0.0115	0.0069
1:100	2	0.0014	0.0012	0.0047	0.0026	0.0093	0.0066
	5	0.0014	0.0012	0.0046	0.0025	0.0092	0.0065
	10	0.0014	0.0012	0.0047	0.0025	0.0095	0.0066
	20	0.0014	0.0012	0.0046	0.0025	0.0093	0.0065

aggravated by the lower sampling rate. Traffic fluctuations at small timescales cannot be estimated from data sampled at such low rates and, therefore, the monitored data is useless for link dimensioning.

To quantify the loss on accuracy of link dimensioning, we sample the traces from the datasets of location B and C at rates 1:10 and 1:100 with E_2. Tables 3.1 and 3.2 present the average and standard deviation of $\hat{\varepsilon}$ (see Equation (3.12)) obtained for all traces from locations B and C, respectively. For these experiments we use Equation (3.9) to estimate the variance from sampled data, and we set $\varepsilon = 0.01$ in the dimensioning formula. The conclusion from these results is that the buffer size BS (in number of packets) does not seem to play a major role in the accuracy of the link dimensioning. Once again, $\hat{\varepsilon}$ is actually influenced by the chosen sampling rate and obtained values are comparable to those in Figure 3.9. For the majority of the cases the link dimensioning is successful, *i.e.,* $\hat{\varepsilon} \leq \varepsilon$. However, it is important to remember that one should be careful on

assuming the link dimensioning was successful when $\hat{\varepsilon}$ equals or is close to zero. As explained before, in Section 3.5.5, when $\hat{\varepsilon}$ is too low or equals to zero, the link dimensioning procedure might have excessively overestimated the actual required capacity. One of the reasons for such overestimation might be very high traffic peaks that the dimensioning approach tries to account for in the estimation.

3.7 Concluding Remarks

In this chapter we demonstrated that it is possible to have accurate estimations of required capacity by using sampled measurement data combined with the dimensioning formula of Equation (1.2). Many traffic measurement tools implement sampling strategies to cope with the ever increasing traffic rates in high-speed links and reduce the amount of measured data to process. Aiming at an easy-to-use approach, we validated the use of sampled data for link dimensioning from the widely available tool sFlow. In addition, we also validated the use of sampled data obtained from two other sampling strategies, namely, Bernoulli and n-in-N sampling.

We demonstrated that accurate estimations are obtained provided that a reasonable sampling rate is chosen taking in to consideration the traffic volume (*i.e.,* load in the monitored link) and the timescale of interest. Inappropriate sampling rate, *i.e.,* too high for the traffic volume, might result in sampled data that do not represent the actual observed traffic. Consequently, traffic properties such as Gaussianity fit might be lost and, ultimately, the link dimensioning procedure might provide inaccurate estimations. This inaccuracy might come in the for of either underestimation or excessive overestimation of required link capacity.

Our results proved that if traffic measurements are done for purposes, among others, of link dimensioning, it is not enough to base the decision on what sampling rate to use solely on the capacity of the monitored link[3]. Instead, one should consider the actual traffic volume and the timescale of interest for link dimensioning. Figure 3.14 shows the sampling rates that were applied to traces from our dataset that had required capacity successfully estimated at timescales from 1ms to 1s. That is, this figure plots the highest sampling rate for each trace that provided enough sampled data to successfully estimate required capacity at different timescales. Notice that in this figure each dot is a trace, and traces are the same for each timescale. Traces do not change in position on y-axis (which is trace's throughput), but they might change in color for different values of timescale on x-axis. In addition, there is no distinction

[3]http://blog.sflow.com/2009/06/sampling-rates.html

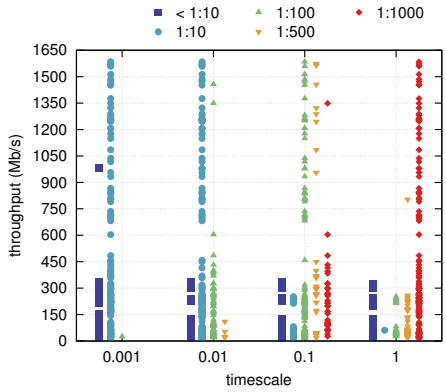


Figure 3.14: Relationship between sampling rate, timescale and traffic volume, using 1-in-N sampling.

between traces' locations in this figure. As expected, Figure 3.14 shows that lower sampling rates predominate at smaller timescales and higher sampling rates predominate at higher timescales.

One clear example in the figure is the trace plotted around 600 Mb/s. For this trace, data sampled 1:10 was enough to successfully estimate required capacity at $T = 1$ms. For the same trace, at $T = 10$ms, the sampling rate 1:100 provided enough data to result in correct estimation of required capacity. At $T \geq 100$ms, however, data sampled 1:1000 for this trace was sufficient to obtain correct estimations. Another important take away from this figure is that many traces with lower traffic throughput actually needed very low sampling rates, *i.e.,* $r < 10$ at any timescale so that sampled data was enough for correct estimations of required capacity. In addition, there was no case among traces from our dataset in that higher sampling rates, *i.e.,* $r > 100$, provided sufficient data for our link dimensioning approach to correctly estimate required capacity.

Results shown in Figure 3.14 provide a good guideline for defining which sampling rate to use. Nonetheless, the definition of a rule of thumb for choosing which sampling rate to use is not trivial. That's because besides traffic volume and timescale, some other traffic characteristics can also play a role in the choice of the sampling rate. For example, if the actual traffic variance is not high, *i.e.,* traffic rates are quite constant, higher sampling rates could potentially be applied even for link dimensioning at smaller timescales. However, this is a deadlock situation. The fact that the traffic variance is not known justifies the need for the traffic measurements and for the link dimensioning approaches proposed in this chapter.

Another important contribution of this chapter is that we proved that the usual simplistic approach of scaling the traffic variance from sampled data, *i.e.,* by the square of the inverse of the sampling rate, results in an excess of the actual traffic variance and, ultimately, leads to excessive overestimation of required capacity. Therefore, we propose and validate approaches to better estimate traffic variance out of sampled data for each one of the sampling algorithms studied in this chapter. To better estimate traffic variance from sampled data, these approaches take into consideration the additional variance added by the sampling procedure.

Finally, we identified potential threats in the implementation of sFlow to the link dimensioning procedure. We assessed whether the buffer size in the sFlow agent and the missing inter-arrival times of sampled packets would impact on the accuracy of the estimations of required capacity. On the one hand, our results showed that, for the range of tested values, the buffer size does not affect the link dimensioning approach. On the other hand, however, we verified that the lack of individual packet timestamps in the sFlow exporting process can invalidate the sampled data for link dimensioning purposes. Nonetheless, higher sampling rates can mitigate these effects.

3.7.1 Practical Considerations for sFlow

Besides the exporting process investigated by us, there are additional technical questions in commercial implementations of sFlow that might impact on the quality of the traffic measurements and, consequently, on the accuracy of the link dimensioning.

sFlow datagrams use UDP as transport layer protocol. According to [95], UDP is more robust than a connection-oriented protocol for this purpose and the only effects on the performance of an overloaded system is a slightly increase in transmission delay and a greater number of packet losses. Although claimed as insignificant, packet loss might drastically reduce accuracy of operations such as link dimensioning. From real-world deployments we have experienced very high numbers of packet loss using the default installation of sFlow in network devices. This might also be caused by running the sFlow agent jobs with a low priority on the exporting router or switch. Therefore, we generally advise to avoid losses of sFlow datagrams as much as possible by giving agent jobs the right priority and by providing enough residual link capacity between the agents and the collectors if they are not physically located on the same host.

The sFlow protocol also defines and exports a counter called *sample_pool*. This counter gives the total number of observed packets before sampling and it can be used to further determine the actual effective sampling rate. This information could support potential improvements on the proposed approaches

for estimating the traffic variance from sampled data, presented in Section 3.4. Therefore, there is room for a follow-up research in order to explore how much this counter can actually contribute to an even better estimation of required capacity.

Finally, it is important to consider that typically operators simply enable the sFlow monitoring in their network devices keeping default configurations. By studying official documents of sFlow-enabled devices[4] we found out that the default settings vary quite a lot from vendor to vendor. For example, sFlow in Dell and IBM devices has sampling disabled by default (*i.e.,* a sampling rate of 1:1). However, the default sampling rate for sFlow in Brocade devices is 1:2048 and for Cisco devices is 1:4096. For the latter cases, the measured traffic is of very limited use, if any, for link dimensioning. If the network operator uses the sampled data for link dimensioning purposes, the default sampling rate should be verified and changed following the intuitive guideline given in Figure 3.14 and the conclusions enabled by the results from experiments in Section 3.5.

[4]http://www.sflow.org/products/network.php

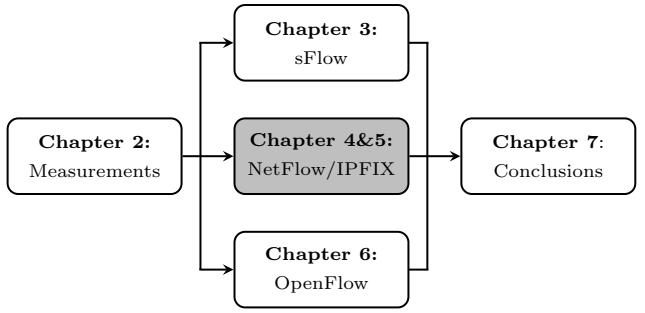
It's not the strongest or the most intelligent who will survive but those who can best manage change.

— Charles Darwin

CHAPTER 4

Pure Flow-based Link Dimensioning

Flow-level traffic measurements have become one of the most ubiquitous technologies in today's networks, mostly because of the widely available Cisco's NetFlow and its variations. Flows provide a summary of the observed traffic and, hence, are an scalable alternative to onerous plain packet capturing. Scalability comes, however, at the cost of information loss: inter-arrival time of packets is abstracted in flows. In particular to the case of this thesis, the lack of timestamps of individual packets make it harder to estimate traffic variance, which is an important parameters of the dimensioning formula of Equation (1.2)*. In this chapter we propose and validate an approach to estimate traffic variance and, ultimately, required link capacity, purely from flows (NetFlow/IPFIX style). The publication related to this chapter is [45].*



The organization of this chapter is as follows:

- *Section 4.1 gives a background on flow-level traffic measurement, focusing on NetFlow and IPFIX, and states the contribution of this chapter.*
- *Section 4.2 introduces the proposed flow-based approach for link dimensioning.*
- *Section 4.3 validates the proposed approach.*
- *Section 4.4 concludes this chapter.*

4.1 Background

In this section we provide a brief background on the history of flow-level traffic measurements focusing on NetFlow and IPFIX. Then, we explain the flow measurement operation. Last, we detail the challenge of using flows for link dimensioning and state the contribution of this chapter.

4.1.1 NetFlow & IPFIX Facts and Figures

One can find different views on the relationship between NetFlow and IPFIX in the literature. It is not a goal of this thesis to take a position in this discussion. Nonetheless, in this section we briefly present the viewpoint of two respected researchers/engineers actively involved in both NetFlow and IPFIX.

The first point of view comes from an interview[1] with Benoit Claise, a Cisco Distinguished Engineer and IETF area director for the operations and management, during the IETF 87 meeting. According to Claise, the history of NetFlow starts in the mid 90's when it was not only a monitoring tool, but its main application was actually as a switching path. It slowly gained popularity as a monitoring tool because operators would resort to measurements from NetFlow when facing problems in their networks. Nowadays the switching path has been replaced with more modern technologies for that specific purpose, such as the Cisco Express Forwarding [24], and NetFlow has focused on traffic measurement.

There are many version of NetFlow export protocol. Among them the most famous are certainly NetFlow v5 [28] and v9 [32], which nowadays are the two available versions for Cisco's customers. NetFlow v2 to v4 are Cisco's internal versions and they have never been released. NetFlow v6 was developed to attend the particular needs of a specific customer and are not supported by Cisco's flow collector. NetFlow v8 [25] implements flow aggregation for information present in NetFlow v5, typically aggregation per IP prefix. Yet according to Claise, NetFlow v5 is the most common version available in Cisco devices and it is still supported by Cisco. Some of the major differences between NetFlow v5 and v9 is the flexibility on the flow definition, enabled by the Flexible NetFlow technology [30], and the possibility of measuring IPv6 traffic or other technologies such as MPLS.

To some, including Claise, IPFIX [34, 35], standard within the IP Flow Information Export (ipfix) WG[2] can be also defined as NetFlow v10. Another viewpoint on the NetFlow vs. IPFIX discussion is given by Brian Trammell, a researcher at ETH Zurich and an active contributor at several IETF groups,

[1] `https://www.youtube.com/watch?v=w61IZ6kYWAQ`. Accessed on Jun. 2014.
[2] `http://datatracker.ietf.org/wg/ipfix/`. Accessed on Jun. 2014.

in his interview[3] also during IRTF 87 meeting. Trammel defines NetFlow as a set of technologies, such as metering processes and export protocol, under the umbrella of a Cisco product, while IPFIX is more focused on the export protocol.

Nonetheless, as mentioned above, it is not our goal to position ourselves on whether and how NetFlow and IPFIX relate. In this thesis we are particularly interested in NetFlow v5 simply by the fact that this protocol already provides the information we need for link dimensioning purposes. In addition, the same information we use from NetFlow v5 can also be obtained from NetFlow v9 and IPFIX protocols. Therefore, the link dimensioning approach proposed in this chapter and the one proposed in Chapter 5 are referred to as flow-based approaches, which means that they make use of NetFlow/IPFIX style flows.

Besides proprietary implementations of NetFlow available on Cisco devices, one can find many open source and independent implementations that do measure, export and process NetFlow/IPFIX-like style flows. A non-exhaustive list of implementations is given in the following:

- YAF (Yet Another Flowmeter) [19, 61] is a flowmeter originally intended to both track the developments and become a reference implementation of IPFIX metering and exporting processes. As later explained in Section 4.3.1, we use YAF to convert the packet captures from our dataset, as described in Section 2.3, into NetFlow-like flows.

- `nfdump`[4] is a set of tools to collect and process NetFlow data. It supports NetFlow v1, v5, v7, v9 and IPFIX.

- `fprobe`[5] is a flow exporter that captures traffic and exports NetFlow flows to a NetFlow collector. It supports NetFlow v1, v5 and v7.

- `pflow`[6] is a OpenBSD implementation of a kernel interface for NetFlow-like flow export. It is compatible with NetFlow v5, v9 and IPFIX.

- `ipt-netflow`[7] (NetFlow iptables module) is a Linux implementation of a NetFlow exporting module, which also supports NetFlow v5, v9 and IPFIX.

Besides NetFlow/IPFIX, there are other flow-level traffic monitoring and measurement technologies. For example, Argus (Audit Record Generation and Utilization System) [98] is a comprehensive set of tools that capture packets,

[3] `https://www.youtube.com/watch?v=bMF3coSAl0s`. Accessed on Jun. 2014.
[4] `http://sourceforge.net/projects/nfdump/`. Accessed on Jun. 2014.
[5] `http://sourceforge.net/projects/fprobe/`. Accessed on Jun. 2014.
[6] `http://www.openbsd.org/cgi-bin/man.cgi?query=pflow`. Accessed on Jun. 2014.
[7] `http://sourceforge.net/projects/ipt-netflow/`. Accessed on Jun. 2014.

generate flows and provide detailed network status reports. Argus defines its own information model and transport. Besides the traditional flow definition, as done by NetFlow, Argus enables operators to define flows at other network layers using different combination of fields. Nonetheless, in this chapter we focus only on NetFlow, which is undoubtedly the most deployed flow-level solution.

4.1.2 Monitoring Traffic at the Flow Level

This section provides a general explanation on how the monitoring of traffic at the flow level works. This explanation is based on the requirements for IPFIX as published in [99]. Before explaining the overall operation of monitoring traffic at the flow level, it is important to understand two basic concepts, namely *flow* and *flow records*. Their definitions are as follows.

> ***Flow*** *is defined as a set of packets that share common properties passing an observation point in the network. A commonly used flow definition is based on the 5-tuple key from NetFlow v5 consisting of source and destination IP addresses, source and destination ports and transport protocol.*

> ***Flow record*** *is defined as a report that contains information about a specific flow. Besides the flow identification (5-tuple) a flow record also contains properties of the flow that were measured during the flow record's lifetime in the metering process (*e.g., *number of bytes and packets). Information of a single flow can be split into multiple flow records.*

Figure 4.1 shows the overall operation of monitoring traffic at the flow level. The observation point is a location in the network where IP packets can be observed. This can be, for example, one or a set of interfaces of a router, a single port of a router or even all interfaces of a line card. Using the packets observed at the observation point as input, the metering process is responsible for a set of operations such as packet header capture, time-stamping, packet sampling and classification. From the treated packets, the metering process generates and maintains flow records. This maintenance consists of, among others, flow record creation, statistics computation and forwarding flow records to the exporting process when these have been expired or completed.

A single flow can be exported as many flow records. What defines it are the timeouts. There are two timeouts, namely active and inactive (or idle) timeout. The *inactive timeout* defines the maximum interval between the last observed packet belonging to the flow and the moment at which the metering process considers the flow as terminated and the current flow record is sent

Figure 4.1: Flow-level traffic monitoring.

to the exporter. That is, flows with packet inter-arrival times larger than the inactive timeout are split into multiple flow records by the metering process. The *active timeout* tells the metering process the maximum duration of a flow record before being sent to the exporter, even if its respective flow is still active (*i.e.,* last observed packet of the flow is within the interval defined by the inactive timeout). Therefore, flows that last longer than the active timeout are also split into multiple flow records.

The received flow records at the exporting process are sent to the collector using the (appropriately) chosen export protocol. Finally, at the collector flow records are either stored or further processed and forwarded to other applications.

Reliability of flow-level measurements

An important aspect of measuring traffic at the flow level is the quality of measured data. In [57] the authors have shown that the flow export process may introduce artifacts (*i.e.,* measurement errors) in the exported data. They also show that this problem is found in devices from various vendors. In this chapter we do not address the quality of the measured flow data. Such study is out of this thesis' scope and, therefore, we assume that exported flow data is artifact-free. Nonetheless, it is important to keep in mind that in real deployments the proposed link dimensioning approach in this chapter might inherit problems originally caused by artifacts in the flow export process.

Random Sampled NetFlow

Random Sampled NetFlow [27] allows for the operator to activate sampling on measuring traffic. The sampling algorithm of Random Sampled NetFlow has been explained in Section 3.3.2. In real-world deployments, we have observed cases in which network operators set very low sampling rates even for traffic rates that do not justify such a choice (*e.g.,* from 1:500 to 1:2048). If sampling rates are very low, the exported data might lose important information, becoming unusable for link dimensioning (as explained in Chapter 3). We have also observed real NetFlow deployments in that operators disable packet sampling for NetFlow measurements. Nevertheless, although sampling is employed in

many cases, we expect to see in the future that operators start to increase the sampling rates, or even disable it, in NetFlow-like monitoring systems. That's because flow-level measurements are also being extensively used to support security operations [55, 56] and, for example, in such operations sampled flow data might result in many false negative problems on attempting to identify malicious traffic. Therefore, in this chapter we do not consider sampled flow data. However, on using sampled flow data, one can find valuable information in the discussion about sampling rates and link dimensioning in Chapter 3.

4.1.3 Challenge & Contribution

Flows can provide more information of the observed traffic than basic SNMP counters. In addition, flow level monitoring is a scalable alternative to plain and continuous packet capturing. However, the data aggregation performed by the flow monitoring comes at the cost of information loss. Typically, a flow record does not contain information on individual packets, such as the packet arrival time and the packet size. That is, it is not possible to correctly track traffic fluctuations from flows. This directly impacts the problem of link dimensioning, since these are important information to compute essential traffic characteristics, *e.g.,* the traffic variance required by the adopted dimensioning formula in this thesis. Therefore, the information loss imposes a challenge on the use of flows for link dimensioning.

Aiming at overcoming the information loss inherited from flows, and to enable their use on link dimensioning, in this chapter we propose a straightforward approach to estimate traffic variance from flows. This approach is based on the assumptions that (i) all packets in a flow are uniformly distributed within the flow duration and (ii) all packets in a flow have constant size. Such assumptions are clearly not realistic. However, we expect that in the presence of a very large number of flows, the averaging error introduced by our assumptions will be alleviated. In this chapter we validate the proposed approach within the complete link dimensioning procedure by quantifying the under and overestimations of required capacity when using flow-level measurements.

It is important to mention that the approach proposed in this chapter aims at cases in which network operators already possess the technology for flow-level traffic measurements (*e.g.,* Cisco's NetFlow). However, since flow data is also being used for other purposes, changes cannot be made on the operating measurement system on behalf of link dimensioning. Nonetheless, we are aware that, concerning flow-level measurements, additional implementations with potential changes in the measurement system, one might be able to solve the problem of missing information in an easier way. This is discussed in more details at the end of this chapter.

4.2 Flow-based Approach

Considering the problem of loss of crucial information in flows, in the context of link dimensioning, in this section we describe our proposed approach for estimating traffic variance from flows and ultimately computing the required link capacity. The whole approach is quite straightforward and it is presented in two subsections. The first one describes the overall structure of the proposed approach. Then, we detail the process for creating traffic time series from flows (aka *flow-level time series*) and using them to estimating traffic variance.

4.2.1 Approach Overview

As already mentioned, the approach in this chapter is quite straightforward and based on general assumptions of traffic. The approach can be divided in three steps, as showed in Figure 4.2. Operations in the first step are actually related to the flow-level traffic monitoring. That is, the first step consists of operations from Figure 4.1, namely, metering and exporting processes. The input flow records in the second step can be either received directly from the exporting process or retrieved from the flow collector. It will depend on the architecture of the implemented system for link dimensioning. The whole link dimensioning is, however, out of this thesis scope. On having flow records the flow-level time series are created in the second step, as detailed in the next section. Finally, in the third section traffic statistics, such as traffic variance, are estimated from the flow-level time series.

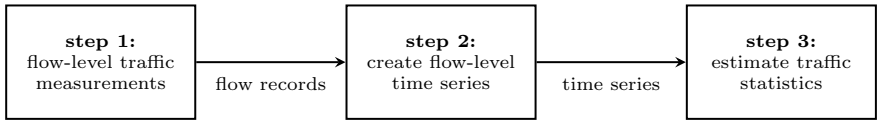


Figure 4.2: Flow-based link dimensioning approach.

4.2.2 Creating Flow-level Time Series

In this section we detail the procedure of creating flow-level time series, which is a sequence of values that represent the link usage (*i.e.,* traffic rate) during the entire measured period. To create flow-level time series, the only information needed from the flow records are: the start time s, the end time e and the number of bytes b. We assume that the bytes belonging to a flow record are uniformly distributed over the flow record's duration. This assumption imposes

another assumption on that packets belonging to a flow record have constant size defined by the flow record's *number of bytes/number of packets*.

Initially, the flow records are placed in a timeline that lasts from s of the first flow record to e of the last flow record in the measurement period. The timeline is defined as

$$T_F = \{B_{F,i}\} \, , 1 \leq i \leq m \, , \tag{4.1}$$

where $B_{F,i}$ is the link usage in the time interval $t_i = [iT, (i+1)T[$, and T is the interval size that defines the granularity for the following computations (*i.e.,* timescale). In order to create the flow-level time series, we have to calculate for each time interval t_i the amount of bytes transferred in that interval. Let's consider a set of flow records F consisting of n records f_i, where $1 \leq j \leq n$. The subset of flow records that contribute to the traffic rate in time interval t_i is given by

$$k_i = \{f_i \in F : s_j < (i+1)T \wedge e_j \geq iT\} \, . \tag{4.2}$$

For each flow record in k_i we calculate the amount of bytes it contributes to the time interval t_i. The sum over all flow records in k_i gives the total number of bytes S_i transferred in t_i

$$S_i = \sum_{f_j \in k_i} \frac{\max(\min((i+1)T, e_j) - \max(iT, s_j), 1)}{\max(e_j - s_j, 1)} \cdot b_j \, . \tag{4.3}$$

The above formula accounts for the fact that flows do not necessarily start or end exactly at the beginning or end of the time interval t_i. It considers the fraction of bytes in relation to the flow record duration within the specific interval. The max function in the divisor avoids a division by zero when the flow record's start time is equal to its end time. This happens when the flow record consists of, for example, only one packet. The flow-based link usage $B_{F,i}$ in t_i is then given by dividing S_i by the size of the chosen timescale

$$B_{F,i} = \frac{S_i}{T} \, . \tag{4.4}$$

By having the sequence of link utilization for all t_i in the measured period, the flow-level time series is complete. The average traffic rate and the traffic variance are then calculated in the traditional way by, respectively

$$\rho = \frac{1}{m} \sum_{i=1}^{m} B_{F,i} \, , \tag{4.5}$$

and

$$\upsilon(T) = \frac{1}{m-1} \sum_{i=1}^{m} (S_i - \rho T)^2 \, . \tag{4.6}$$

Finally, the calculated ρ and $\upsilon(T)$ are applied to the link dimensioning formula of Equation (1.2). What is ultimately obtained is the estimation of required capacity $C(T, \varepsilon)$ obtained purely from flow-level traffic measurements. In the next section we validate the proposed approach using the traffic measurements from the dataset introduced in Chapter 2.

4.3 Experimental Results

In this section we validate the proposed approach in this chapter. We assess the accuracy of the link dimensioning approach using traffic statistics computed from flow-level measurements. Given that our dataset, as presented in Chapter 2, consists entirely of packet captures, we first need to convert those into flows. This procedure is described in Section 4.3.1.

The analysis and discussion of our results is divided in two parts. The first part, in Section 4.3.2, consists of a qualitative analysis by means of manual comparison of the flow-level time series with a "ground-truth" time series created from the packet-level traces. This analysis aims at identifying situations in which flow-level time series show a different behavior than the packet-level one, and to investigate the causes of deviations. The second part, in Section 4.3.3, consists of a quantitative analysis in which we assess the accuracy of the link dimensioning procedure when using traffic statistics computed from flow measurements. The parameters used in the quantitative analysis are the same as the ones introduced in Section 3.5.1. To recapitulate, these parameters are the obtained exceedance probability $\hat{\varepsilon}$ to quantify the underestimation of required capacity, as defined in Equation (3.12), and the relative error RE to quantify the overestimation of the required capacity, as defined in Equation (3.13).

4.3.1 From Packets to Flows

In order to have flows-level traffic measurements, we converted all packet traces from our dataset into flows. To do so, we used YAF [19, 61], which works both online and offline. In an online fashion, YAF captures live stream traffic and exports flow records. In the offline mode, YAF reads packet captures from trace files, generates and exports flow records following the specified parameters. In our case parameters were the active and inactive timeouts. We have used three combinations of timeouts. That is, from each packet trace we have created three flow traces. Each flow trace from a single packet trace differs in the amount of flow records. Clearly, the shorter the timeouts the bigger the amount of flow records.

The three combinations of timeouts we have defined are presented in Table 4.1. Timeout definitions *a60i20* and *a120i30* are easily found in traffic

Table 4.1: Flow timeout definitions.

abbr.	**timeouts in seconds**	
	active	*inactive*
a5i2	5	2
a60i20	60	20
a120i30	120	30

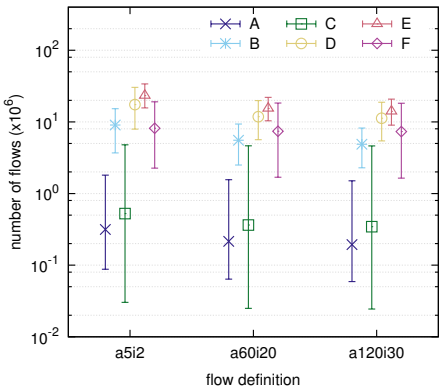
measurement setups at network operators. The definition with shortest timeouts *a5i2*, however, is more difficult to find. That's because setups with short timeout values might confront with infrastructure limitations. That is, on the one hand, short timeouts mean that more measurement traffic will be generated between the flow exporter and the collector (considering a distributed flow monitoring system). One the other hand, however, very long inactive timeouts might create the problem of measurement-related traffic bursts from exporting process to the collector, due to many flow records expiring at the same time. Nonetheless, we have defined *a5i2* serves as a good comparison parameters in the next section.

Flow-level Traffic Characteristics

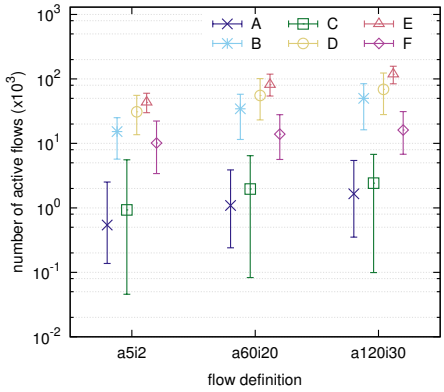
Section 2.3.2 shows the number of packets and average rate of all packet traces in our dataset. Now, Figure 4.3 shows the flow-level characteristics of our traffic for each of the timeouts definition.

Figure 4.3a shows the average number of flow records (and the standard deviation in error bars) for all traces in our dataset for each defined combination of timeouts. As expected, considering the link capacity and utilization (presented in Figure 2.2a), with the *a5i2* flow definition traces from D and E generated two orders of magnitude more flow records than, *e.g.,* traces from A. Another take away from this figure is that, for any location, the small difference between the number of flow records for any combination of timeouts indicates that most of the flows have a duration lower than 5 seconds. For *a5i2*, we can observe a slight increase in the number of flows for locations A and C.

Figure 4.3b shows the average number of simultaneously active flow records per second per trace (and standard deviation in error bars). That is, the average amount of flow records simultaneously being metered every second for all traces of each location. Traces from A have an average of 542 active *a5i2* flow records per second and around 1.6k active *a120i30* records per second. Traces from D and E have an average of, respectively, 30.9k and 48.7k active *a5i2* flow records per second, and around 68.8k and 136.9k active *a120i30* records per second.



(a) Average, max and min (error bars) number of flow records for all traces per location



(b) Average, max and min (error bars) number of simultaneously active flow records every second per location

Figure 4.3: Flow-level traffic characteristics of all traces in our dataset (note the difference on scaling of y-axis).

The longer the timeouts the longer flow records take to be exported by the flow exporter. For example, an inappropriately finished flow (*e.g.,* TCP connection that was not properly concluded), will remain in the metering process until its inactive timeout expires. This explains the increasing number of simultaneous active flows for longer timeouts, as observed in Figure 4.3b. Therefore, although resulting in a smaller number of flow records to be further processed, longer timeouts might demand more resources from the measurement device.

4.3.2 Qualitative Analysis

As mentioned before, flow-level time series introduce averaging errors due to its too optimistic assumption that bytes within flow records are uniformly distributed. In this section we qualitatively check the problems arising as consequence of this averaging. To do so, we manually compare flow-level and packet-level traffic time series.

Figure 4.4 illustrates the differences between packet-level and flow-level time series. This trace has an extreme example that makes it easier to identify averaging problems in flow-level time series. Note that differently of our traces in the dataset, this example trace is 5-minute long. This makes it easier to observe traffic fluctuations. In this example, the situation between 5 and 120 seconds illustrates the problems of averaging when setting long timeouts. During this

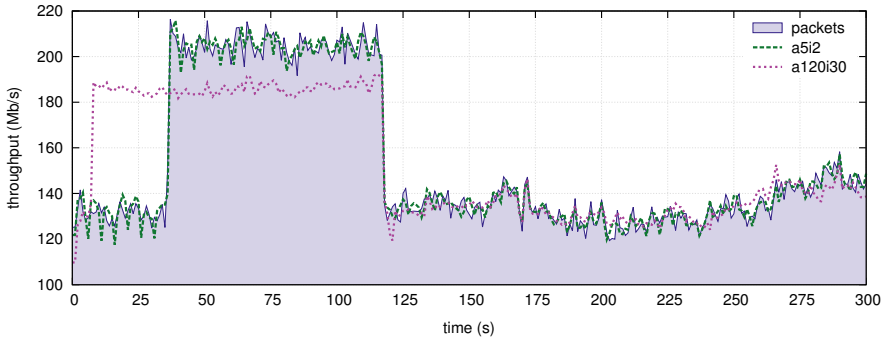


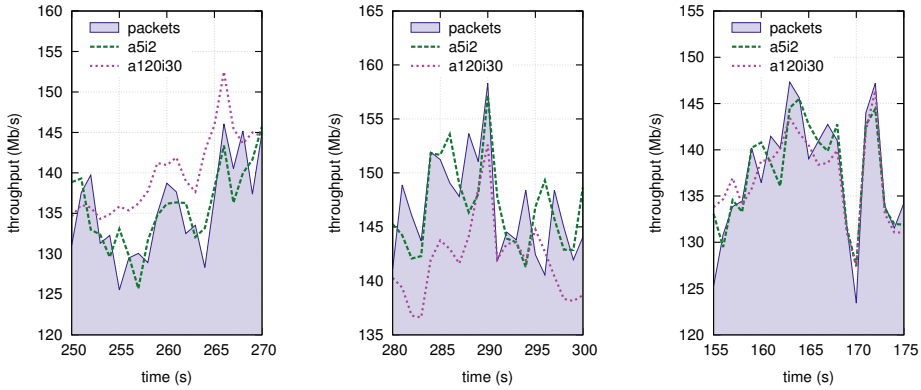
Figure 4.4: Packet-level and flow-level time series at $T = 1$s of a 5-minute long trace.

period, most of the traffic volume was generated by a single flow that was reported as a single flow record when using *a120i30*. As one can see from the packet-level time series, the traffic increased significantly at around 30 seconds. That is, the major portion of bytes was transmitted few seconds after the beginning of the flow. Ignoring the actual distribution and assuming uniformity, our approach averages all the bytes within the entire flow duration.

Comparing the two flow time series, we observe that the *a5i2* series closely follows traffic fluctuations. The reason is that, unlike *a120i30*, the *a5i2* definition exports long flows as several consecutive and smaller flow records. It means that the averaging of traffic is limited to intervals of 5 seconds at maximum (*i.e.,* the active timeout).

In the time series in Figure 4.4 we can identify many matches and mismatches between packet-level and flow-level time series. Particularly, around 250 and 280 seconds we can identify problems in the flow-level time series due to averaging. However, in some intervals such as around 160 seconds, we can also see moments that the flow-level time series nicely follow the packet-level one. These three moments are shown in isolation in Figure 4.5.

In the period zoomed in Figure 4.5a, there is a drop in the traffic rate for a few seconds. The flow-level time series do not follow this drop because they take into account bytes of flows that started before and ended after this specific time bin. As can be seen in the figure, this effect is stronger for larger timeouts. In the period zoomed in Figure 4.5b the problem of averaging can also be clearly seen. The situation that resulted in this mismatch is the same that created the averaging problem observed between 5 and 120 seconds in Figure 4.4. That is, traffic bursts belong to a flow with non-uniformly distributed bytes. Flow records defined with shorter timeouts manage to reflect better fluctuations that



(a) Averaging problem (b) Averaging problem (c) Matching

Figure 4.5: Zoomed intervals of the time series from Figure 4.4.

happen within the period of the timeout (in this case 5 seconds). However, with longer timeouts, a single flow record reports the traffic of the entire flow and, due to our assumptions, the approach believes all bytes are uniformly distributed within the flow duration. Consequently, the time series overlooks traffic bursts that are located in specific intervals within the flow.

Figure 4.5c shows a zoomed interval in which the flow-level time series remarkably follow the fluctuations occurring at the packet-level. This happens when flows have a constant traffic rate, *i.e.,* the flow records represent well the packet-level dynamics. Moreover, this might also happen when flows are short (including flows consisting of a single packet) and, consequently, they fit into one time interval.

We expect the averaging problems in flow-level time series to ultimately impact the obtained estimation of required capacity. In the next section we quantify this impact by assessing the accuracy of our proposed flow-level link dimensioning approach.

4.3.3 Quantitative Analysis

In this section we present the results of extensive validation of the proposed link dimensioning approach. For all traces, after their respective conversion to flow-level measurements as explained in Section 4.3.1, we have used the approach described in Section 4.2 to estimate the traffic variance from flows.

Overall validation

To assess the accuracy of the proposed approach, we quantify the obtained exceedance probability $\hat{\varepsilon}$ as defined in Equation (3.12).

> ***Recap:*** *The estimation of required capacity is considered successful when it yields $C(T,\varepsilon) \geq C_{emp}(T,\varepsilon)$, which is by definition the same as $\hat{\varepsilon} \leq \varepsilon$.*

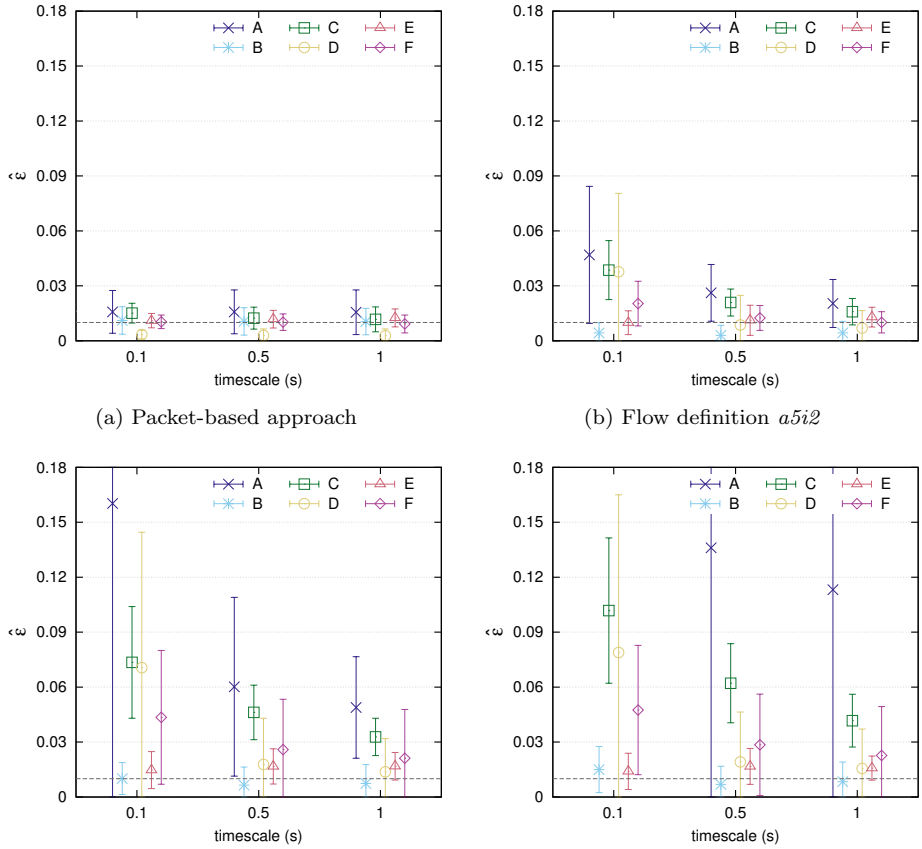
The plots in Figure 4.6 show the average $\hat{\varepsilon}$ obtained for all traces per location for the three different flow definitions. For these experiments, we have set $\varepsilon = 0.01$ (*i.e.,* 1%) in the dimensioning formula of Equation (1.2). Note that the dashed line at the plots of Figure 4.6 highlight the case of $\hat{\varepsilon} = \varepsilon$. For matters of comparison, Figure 4.6a shows the results for a packet-based approach, as originally proposed in [109]. That is, these results were obtained from computations of required capacity using the packet-level traffic traces.

As one can see, the link dimensioning approach purely based on flows does not successfully capture traffic fluctuations that happen at shorter timescales. As a consequence, estimations of required capacity at shorter timescales do not achieve the desired $\hat{\varepsilon} \leq \varepsilon$. Only with the flow definition *a5i2* the average $\hat{\varepsilon}$ was kept within reasonable bounds for $T \geq 500$ms. For the other definitions, however, accuracy of the estimations was compromised even at $T = 1$s.

For traces from locations A and C, the averaging introduced by the link dimensioning is aggravated by the lower amount of flow records per trace, and estimations result in $\hat{\varepsilon}$ that are much higher than the desired ε. This problem gets worse with flow records defined with longer timeouts. (Note that in Figure 4.6d, for flows *a120i30*, the average $\hat{\varepsilon}$ for traces from location A at $T = 100$ms is not even plotted because it is higher than the y-axis upper bound.) However, for traces from locations B, D and E estimations of required capacity were accurate even at $T = 100$ms and for any flow definition. The higher number of flow records per trace in these locations, for any of the considered flow definitions, as shown in Figure 4.3a, helps to alleviate the impact of the averaging problem introduced by the proposed link dimensioning approach.

In fact, for traces from these three locations, good accuracy of estimated required capacity was obtained even at much shorter timescales. The additional plot in Figure 4.7 shows that the average obtained $\hat{\varepsilon}$ for locations was satisfactory at timescales as short as $T = 25$ms. One important remark is that traces from location D can be divided in two groups: those with higher average rates and those with lower average rates. In Figure 4.7 only the former ones are plotted, and they account for half of all traces from D.

Therefore, we can conclude that the proposed approach can provide accurate estimations of required capacity at timescales as low as 25ms, provided that



(a) Packet-based approach

(b) Flow definition *a5i2*

(c) Flow definition *a60i20*

(d) Flow definition *a120i30*

Figure 4.6: Average and standard deviation (error bars) of $\hat{\epsilon}$ at various T.

the measured traffic has a high enough number of flow records (*e.g.*, in our dataset, location E has an average of around 14M *a120i30*-type flows per 15-minute trace). However, if the monitored link has an average number of flow records similar to those from the other locations in our dataset, it is likely that estimations will mostly underestimate the actual required capacity if flow records are defined with longer timeouts.

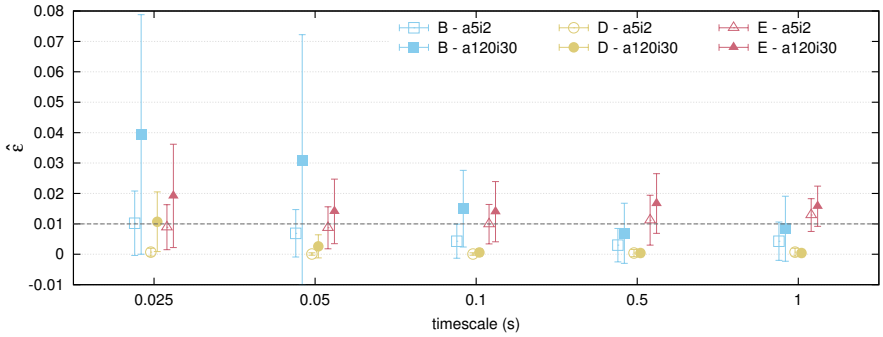


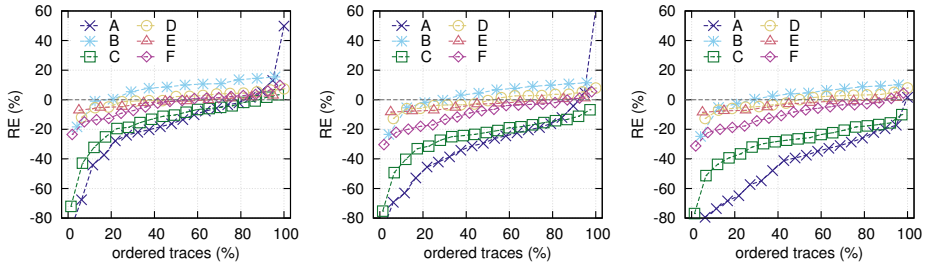
Figure 4.7: Average and standard deviation (error bars) of $\hat{\varepsilon}$ at various T for traces from B, D and E.

Quantifying the underestimation

From the results in Figure 4.6, we learn that the longer the flow timeouts and the shorter the timescale, the lower the accuracy of the estimations using the proposed flow-based approach. However, it is also important to understand how far is the estimated $C(T, \varepsilon)$ from the actually required $C_{emp}(T, \varepsilon)$. The plots of Figures 4.8 and 4.9 show the relative error RE – given by Equation (3.13) – for $T = 500$ms and $T = 1$s, respectively. It is important to remember that in these plots, the value of RE=0% (highlighted with a dashed line) corresponds to $C(T, \varepsilon) = C_{emp}(T, \varepsilon)$.

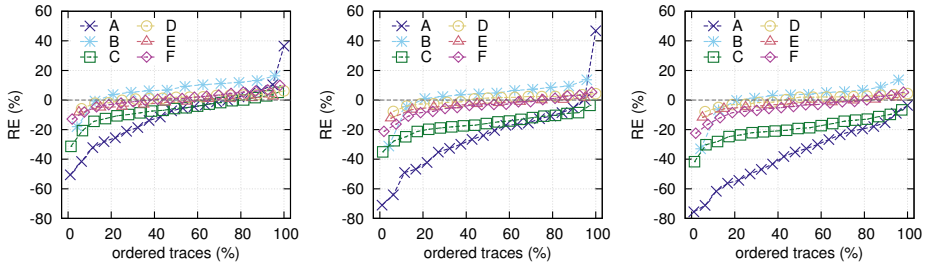
> ***Recap:*** *The relative error RE is used to assess how much for more (overestimation) or for less (underestimation) the estimated required capacity $C(T, \varepsilon)$ differs from the empirical ground truth $C_{emp}(T, \varepsilon)$.*

From the plots in these figures we can see that underestimation is the main problem with the proposed flow-based approach. On the one hand, in the worst cases, for locations A and C at $T = 500$ms, the estimated $C(T, \varepsilon)$ is, for few traces, more than 80% lower than the required $C_{emp}(T, \varepsilon)$. This explains why in Figure 4.6d the average $\hat{\varepsilon}$ is not plotted within the defined range for y-axis. The underestimation for these traces, mainly from those from C, is alleviated at $T = 1$s. Only few traces from location E had their capacity estimated around 30% less than the required as defined by the empirical estimation. For traces from locations B, D and E, on the other hand, underestimation was kept within reasonable bounds. Even for flows *a120i30* the estimated $C(T, \varepsilon)$ is at most about 15% less than the required $C_{emp}(T, \varepsilon)$.



(a) Flow definition *a5i2* (b) Flow definition *a60i20* (c) Flow definition *a120i30*

Figure 4.8: Relative error RE for all traces per location at $T = 500$ms.



(a) Flow definition *a5i2* (b) Flow definition *a60i20* (c) Flow definition *a120i30*

Figure 4.9: Relative error RE for all traces per location at $T = 1$s.

From the results in Figures 4.8c and 4.9c, for the cases of longer flow timeouts, the estimations of required capacity obtained from the proposed flow-based approach can serve as a good baseline estimation. That is, in practice the operator could still need to use a rule of thumb and add a safety margin on top of the obtained estimation. However, instead of 30% of the average rate, between 5% and 10% would suffice for most of the traces from our dataset, but those from A and C. Such margin would help to approximate the estimation of required capacity to the empirical one, consequently, alleviating the underestimation problem.

4.4 Concluding Remarks

Flows are a scalable alternative to continuous packet capturing for measuring traffic because they provide a summarized overview of the observed traffic. However, the scalability comes at the cost of inherited information loss. The

individual timestamps and sizes of observed packets is not recorded. Consequently, the information of how packets and bytes are distributed within the flow is unknown, which makes it difficult to calculate the traffic variance later used in the dimensioning formula.

In this chapter we have proposed a link dimensioning approach purely based on flow-level traffic measurements. This approach is built upon the (unrealistic) assumption that bytes transferred during the flow are uniformly distributed within the flow's duration. Although this assumption is unrealistic, we have proved that this simplistic approach can lead to accurate estimations of required capacity relying solely on flows.

The accuracy, however, is conditional on the number of flows that amount to the measured traffic aggregate and the average duration of these flows. The best results were achieved for traces from locations B, D and E, whose traces have an average of between 5 to 14 million flows (defined as *a120i30*). The worst results were for locations A and C, whose traces have an average of between 1.6 and 2.4 thousand flows (defined as *a120i30*).

The definition of timeouts and, respectively, a upper bound for flow duration, also impacts on the accuracy of estimations. We have demonstrated that, on the one hand, the shorter the timeouts, the better our procedure reconstructs short-term traffic fluctuations. On the other hand, longer timeouts aggravate the averaging introduced by our approach due to the assumption of uniformly distributed bytes within flows records. We have used three timeouts combinations in our experiments. In addition, we observed that for some locations the accuracy was not significantly compromised even with longer timescales. For example, estimations for traces from location E achieved good accuracy using *a120i30* flows. By analyzing Figure 4.3a we can assume that such good accuracy was possible because most of flows from location E are shorter than 5s. Therefore, in such cases the definition of longer timeouts does not impact significantly on the estimation of required capacity.

To summarize, the link dimensioning approach proposed in this chapter is able to estimate required link capacity for large timescales, *e.g.,* 1s. However, as long as the measured traffic consists of a "high enough" number of flows, the proposed approach is able to provide accurate estimations of required capacity at timescales as low as 25ms. That's because a high number of flows makes up for the averaging introduced by our approach. On having "too few" flows, our assumption (of uniformly distributed bytes within flows) overlooks the traffic fluctuations at short timescales and, ultimately, leads to inaccurate estimations mainly at shorter timescales. This problem can be partially palliated by using shorter timeouts.

Therefore, the pure flow-based link dimensioning approach can be used by operators targeting QoS for basic user operations, such as web browsing, for

which estimations at $T = 1$s suffice. However, if the operator wants to guarantee QoS for more demanding services, such as online video streaming, shorter timescales should be considered. In Chapter 5 we propose a hybrid link dimensioning procedure that combine flows with other strategies aiming at accurate estimations of required capacity at much lower timescales, *e.g.,* $T = 1$ms. Finally, a potential extension of the proposed flow-based approach is to identify whether distribution of traffic inside flows changes accordingly to different properties, such as transport protocol and flow duration. If significant differences are found, improvements can be expected by refining the uniformity assumption depending on such properties.

4.4.1 Practical Considerations

We believe that an operator is not going to change or deploy the entire monitoring system only with the purpose of having extra information for link dimensioning. Therefore, the link dimensioning approach proposed in this chapter focuses on cases where operators cannot make additions and changes to their current traffic monitoring systems. For example, parameters definition in the monitoring system cannot be changed because flows are also used by other, and sometimes critical, operations. Another situation is that operators rely on data provided by the monitoring system embedded in their network devices. That is, if the router supports only traditional NetFlow metering process and NetFlow v5, the monitoring system becomes limited to what the technology offers and the operator is unable to implement additions on it.

There might be easier ways to solve the problem of link dimensioning. For instance, one can have good estimations of traffic variance if the monitoring system allows for additional traffic statistics to be coded and added to the metering process (*e.g.,* new information elements in NetFlow v9 or IPFIX). Ultimately, the more accurate the estimation of traffic variance, the more accurate the estimation of required capacity. In addition, if the whole traffic aggregate can be monitored as a single flow (this would require a change from the common 5-tuple flow key definition to a wildcard definition), flow records would report statistics about the whole traffic aggregate and not in a per-flow basis. Now, if the additional statistic to be metered for the whole aggregate is traffic variance, the exported information could directly be used in the link dimensioning formula to obtain the estimation of required capacity. This way, approaches such as the one proposed in this chapter would not be needed. However, this kind of modification in the monitoring system would only be possible in cases that the system is deployed to mainly support link dimensioning. Other approaches depending on the monitoring system would need to adapt to the changes. Clearly, this is a quite unrealistic scenario.

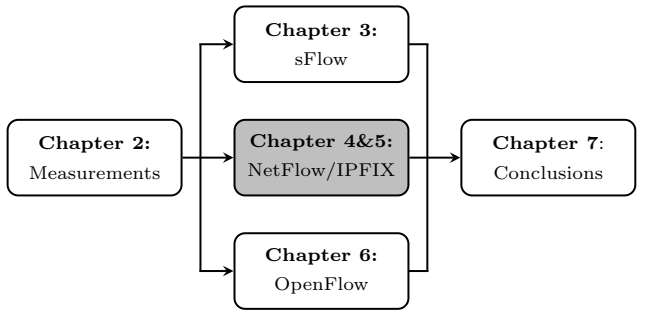
By all means let's be open-minded, but not so open-minded that our brains drop out.

— Richard Dawkins

CHAPTER 5

Hybrid Flow-based Link Dimensioning

In the previous chapter we proposed a pure flow-based link dimensioning approach. Due to the optimistic assumptions on the packet-level traffic properties accurate estimations of required capacity for that approach are limited to higher timescales. In this chapter we propose and validate a hybrid approach for link dimensioning. Supported by flow-level traffic models and sporadic packet capturing, this hybrid approach is capable of providing accurate estimations of required capacity at much smaller timescales, such as 1ms. The publication related to this chapter is [44].



This chapter is organized as follows:

- *Section 5.1 states the motivation and contribution of this chapter.*
- *Section 5.2 sets out the proposed hybrid traffic model.*
- *Section 5.3 describes the procedure for classifying flow records according to their respective duration and rate.*
- *Section 5.4 presents an overview of the complete link dimensioning procedure proposed in this chapter.*
- *Section 5.5 validates the proposed approach for link dimensioning.*
- *Section 5.6 discusses several operational constraints and guidelines on parameters setting for the proposed procedure.*
- *Section 5.7 concludes this chapter.*

5.1 Motivation & Challenges

In Chapter 4 we have proposed an easy-to-use pure flow-based approach to estimate traffic variance out of flow records and ultimately to compute required capacity using the dimensioning formula from [109] – see Equation (1.2). The approach in Chapter 4 assumes bytes to be uniformly distributed within their respective flow record's duration. Such assumption aggravates the averaging problem already existent in flows. That is, short-term traffic bursts might be completely overlooked. Consequently, the calculated traffic variance does not reflect the actual traffic variance as observed at the packet-level and this results in underestimation of required capacity. For the case of traces in our dataset, accurate estimations using the pure flow-based approach were limited to higher timescales. While such large timescale suffices for providing QoS for users on, *e.g.,* web browsing, these are not short enough for more demanding services, such as real-time video streaming.

In this chapter we propose a hybrid link dimensioning procedure that uses flow-level traffic measurements (NetFlow/IPFIX style) combined with analytical models and sporadic packet captures to efficiently describe packet behavior within flows and, hence, accurately estimate required capacity at timescales as short as $T = 1$ms. The traffic model proposed in this paper extends the original model in [112] and allows us to predict traffic variance from flows at small timescales. This variance is then used in the dimensioning formula from [109].

5.2 Models Definition

In this section we detail the flow-level traffic model upon which we build the proposed procedure in this chapter. We start by giving an overview on the base model from [112], and our proposed approach to estimate traffic variance. Then, we present the packet correction factors that use information extracted from packet-level traffic captures to support the modeling of packets behavior within flows.

5.2.1 Flow-based Model

The authors of [112] present an M/G/∞ model to estimate traffic variance $v(T)$ at the flow-level. In its simplest form, the model assumes that traffic flows are created according to a Poisson process with rate λ and have i.i.d. duration D. Furthermore, it assumes that all flows have an identical and constant traffic rate r. The mean throughput is then $\rho = \lambda\delta r$ with $\delta = \mathbb{E}[D]$ and the amount of traffic in a period of time T is $A(T) = r\int_0^T N(t)dt$ with $N(t)$ being the number of active flows at time t.

The basic idea of the model is that $N(t)$ is identical to the number of busy servers in a M/G/∞ queueing station with arrival rate λ and service time distribution F_D. Using this assumption, the variance $v_{flow}(T)$ of $A(T)$ is found to be given by:

$$v_{flow}(T) = \lambda r^2 \Bigg(2T \int_0^T x(1 - F_D(x))dx - \delta \int_0^T x^2 f_{D^r}(x)dx + \delta T^2 (1 - F_{D^r}(T)) \Bigg) , \tag{5.1}$$

where D^r is the residual distribution of D, *i.e.,* $1 - F_D(x) = \delta f_{D^r}(x)$ [112]. As usual, f_X and F_X denote, respectively, the density and distribution function of a random variable X. Knowing the variance, Equation (1.2) from [109], is used to compute the bandwidth requirement $C(T, \varepsilon)$.

5.2.2 Flow-level Traffic Variance

The authors of [112] also give explicit expressions for the variance in case of negative exponentially and Pareto distributed flow durations. For the former the variance becomes:

$$v_{exp}(T) = 2\rho\delta^2 r(e^{-T/\delta} - 1 + T/\delta) . \tag{5.2}$$

By examining empirical data we have found out that the distribution of flow duration is long-tailed and fits better to Pareto- or Weibull-like distributions. Nonetheless, as the authors point out in [112], and we also show in the experiments in this chapter (Section 5.5), the choice of the duration distribution does not affect much the resulting estimated variance. Therefore, one might even consider to use a simpler model, where flows are assumed to have a constant duration δ (further motivations for such a choice will be given in section 5.3). Assuming a deterministic distribution F_D, Equation (5.1) simplifies to (the proof is given in Appendix B)

$$v_{const}(T) = \begin{cases} \rho r(T^2 - \frac{T^3}{3\delta}), & \text{if } T < \delta \\ \rho r(T\delta - \frac{\delta^2}{3}), & \text{if } T \geq \delta . \end{cases} \tag{5.3}$$

5.2.3 Packet-level Modeling of Flows

The basic model in [112] assumes that the traffic rate inside a flow is constant. In general this is not true because IP traffic is transported in form of discrete packets with non-constant inter-arrival times. As a consequence, the basic model underestimates the traffic variance due to possible bursts of packets in a flow.

In [112], the authors also proposed an extension of the model aiming at modeling also the packet details within flows. Assuming that flows consist of packets of constant size s arriving according to a Poisson process, the estimation of the variance becomes (called corrected variance in the following):

$$v_{corr}(T) = v_{flow}(T) + \phi\,. \tag{5.4}$$

with the correction term ϕ given by:

$$\phi_1 = \rho s T \tag{5.5}$$

accounting for the quantized nature of the traffic.

However, our experiments with empirical data reveal that the corrected variance using Equation (5.5) also underestimates the real variance. Therefore, we propose two further extensions of the model by relaxing the assumptions of Poisson arrivals and constant packet sizes. Next, these extensions are detailed.

Poisson arrival and non-constant packet size

It is clear that IP packets are not constant in size. Under the assumption that packet arrivals inside a flow are Poisson distributed with i.i.d. non-constant packet sizes S, the correction term ϕ in Equation (5.4) becomes

$$\phi_2 = \rho T \chi\,, \tag{5.6}$$

where $\chi = \frac{\mathbb{E}[S^2]}{\mathbb{E}[S]}$ with $\mathbb{E}[S]$ and $\mathbb{E}[S^2]$ being the first and second moment of the packet size, respectively. A proof for Equation (5.6) is given in Appendix C.1. Note that Equation (5.5) immediately follows from Equation (5.6) for a deterministic packet size distribution.

Bursty arrival and non-constant packet size

Similar to the previous extension, we assume that the packet size S is not constant. In addition, we assume that packets arrive in bursts of P packets and the time between bursts is i.i.d. and exponentially distributed, where P is geometrically distributed with success probability p, *i.e.,* $\mathbb{P}[P = i] = (1-p)^{i-1}p$. Hence, the packet inter-arrival time IA is hyper-exponentially distributed with squared coefficient of variation $c_{IA}^2 = \frac{2-p}{p}$, which suggests that p can be estimated from an empirically measured squared coefficient of variation by

$$p = \frac{2}{1 + c_{IA}^2}\,. \tag{5.7}$$

Remarkably, a packet burst of P packets can simply be modeled as a huge "super-packet" of byte size $S' = \sum_{i=1}^{P} S_i$, where S_i is the size of the ith packet in the burst, i.i.d. like S. Since P and S_i are independent, we obtain (see proof in Appendix C.2):

$$\mathbb{E}[S'] = \mathbb{E}[S]/p \ ,$$

and

$$\mathbb{E}[S'^2] = \frac{p\mathbb{E}[S^2] + 2(1-p)\mathbb{E}[S]^2}{p^2} \ .$$

Applying this result to Equation (5.6) with $\chi = \frac{\mathbb{E}[S'^2]}{\mathbb{E}[S']}$, the correction term ϕ in Equation (5.4) becomes

$$\phi_3 = \rho T \frac{p\mathbb{E}[S^2] + 2(1-p)\mathbb{E}[S]^2}{p\mathbb{E}[S]} \ . \tag{5.8}$$

5.3 Flow Classification

From the work in [112], we learn that flow rate plays an important role on the calculation of the traffic variance. In addition, from testing the model with empirical data we have observed that using a single set of model parameters for all flows in a measurement period does not provide satisfying results. One reason is the fact that different applications may result in distinct flow characteristics. Another reason is that we are working with flow records, which introduces an artificial upper limit to the flow duration. In order to better account for this behavior, we group flows according to their rate and duration. Ultimately, the traffic variance that goes into the formula of Equation (1.2) is obtained by simply adding up the individual variances of all classes.

Figure 5.1 illustrates how flow records related to each other by their respective rate and duration. This figure shows the positioning of flows in a scatter plot by their rate and duration. We can also clearly see the upper limit introduced to the flow duration given the use of flow records. In this 2-dimensional classification, we divide the rate-duration space into cells of size $\theta \times \eta$ and assign all flow records in the measurement period to flow classes $\Gamma_{ij}, i, j \in \mathbb{N}$, where Γ_{ij} contains all flow records with a traffic rate in the interval $[i\theta, (i+1)\theta[$ and a duration in the interval $[j\eta, (j+1)\eta[$. As done for the classification per rate, for each class Γ_{ij} we determine the flow arrival rate λ_{ij}, the flow traffic rate r_{ij}, the average flow duration δ_{ij}, and the average packet size s_{ij}. On defining a small η as compared to the average duration of flow records, we can assume constant duration δ_{ij} within classes. In this case, δ_{ij} is set to the average duration of flow records in the class Γ_{ij} and Equation (5.3) is used to calculate the flow-level traffic variance for each individual class.

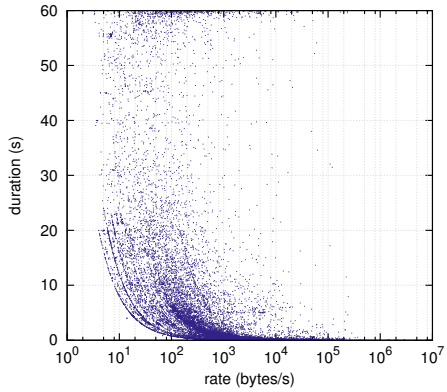


Figure 5.1: Example of flow records relationship by rate and duration; duration upper-bound is defined by active timeout of 60s; for clarity points are sampled every 100 and x-axis is truncated.

5.4 Overview of the Proposed Procedure

The complete procedure to calculate $C(T, \varepsilon)$ from flow record measurements for a given timescale T and bandwidth exceeding probability ε is summarized in Figure 5.2. In this section we describe the procedure using the flows classification per rate θ and duration η. The exact same procedure can be used for classification only by rate. To do so, the value of η should be set to ∞.

The first step (line 1) consists of collecting flow record data for a desired duration M. As explained in Section 4.1.2, flow records depend on the active timeout t_a and the inactive timeout t_i. We will discuss the effects of the timeouts in the experiments in Section 5.5.

In line 2, we assign the flow records to classes according to their traffic rate and their duration. The granularity of the classes depends on the parameters η for the flow duration and θ for the traffic rate. We discuss the use of different combinations of values for η and θ in Section 5.6. Once all flow records have been assigned, the model parameters are determined for each class (lines 4 to 7) and the variance $v_{corr,ij}(T)$ is computed using Equation (5.4). As already explained in Section 5.2.1, the calculation of the variance $v_{flow}(T)$ can be adapted depending on the flow duration distribution (*e.g.,* Exponential or Pareto) or if constant flow duration is assumed. The calculation of the packet correction factor ϕ can also be adapted according to the operators requirements (as discussed in Section 5.2.3).

in: active timeout t_a, inactive timeout t_i
in: measurement duration M
in: timescale T, bandwidth exceeding probability ε
in: flow duration interval size η
in: flow traffic rate interval size θ
out: estimated bandwidth requirement $C(T, \varepsilon)$

1: create/collect flow records with timeout t_a and t_i in a measurement period of length M
2: assign flow records with traffic rate $\in [i\theta, (i+1)\theta[$ and duration $\in [j\eta, (j+1)\eta[$ to class $\Gamma_{ij}, i, j \in \mathbb{N}$
3: **for each** class $\Gamma_{ij}, i, j \in \mathbb{N}$ with flow records $f_1^{ij}, \ldots, f_{N_{ij}}^{ij}$ **do**
4: flow arrival rate $\lambda_{ij} := N_{ij}/M$
5: flow traffic rate $r_{ij} := 1/N_{ij} \sum_{k=1}^{N_{ij}} b(f_k^{ij})/d(f_k^{ij})$
6: flow duration $\delta_{ij} := 1/N_{ij} \sum_{k=1}^{N_{ij}} d(f_k^{ij})$
7: packet size $s_{ij} := \sum_{k=1}^{N_{ij}} b(f_k^{ij}) / \sum_{k=1}^{N_{ij}} p(f_i^{ij})$
8: calculate $v_{corr,ij}(T)$ (see Equation (5.4))
9: **end for**
10: $\rho := \sum_{i,j} \lambda_{ij} \delta_{ij} r_{ij}$
11: $v_{corr}(T) := \sum_{i,j} v_{corr,ij}(T)$
12: $C(T, \varepsilon) := \rho + \frac{1}{T} \sqrt{-2 \log(\varepsilon) \cdot v_{corr}(T)}$

Figure 5.2: Procedure for the estimation of the bandwidth requirement from flow records.

Finally, the overall traffic rate ρ and variance $v_{corr}(T)$ are computed in lines 10 and 11 and the formula of Equation (1.2) is used to calculate the required capacity $C(T, \varepsilon)$ (line 12). Based on the results of our experiments, the selection of values for the parameters t_a, t_i, M, η, θ, T, and ε will be discussed in Section 5.6.

It is important to mention that we disregard flow records with a duration of 0 seconds, which are mostly composed by single packets, because their traffic rate is undefined. Depending on timeouts configuration, these records may account for more than half of all flow records. However, the impact of removing such flows on the proposed link dimensioning procedure is negligible. That is because the total number of bytes from these flows is insignificant as compared to the whole aggregate.

5.5 Experimental Results

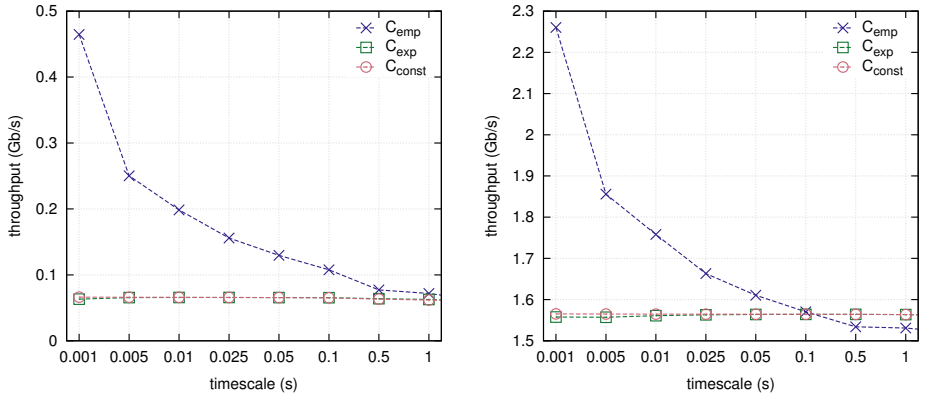
In this section we present and discuss results of experiments with the proposed hybrid flow-based procedure for link dimensioning. The approach used to validate the correctness of the proposed procedure is the same as the one presented in Section 3.5.1.

The content of this section is organized as follows. In Section 5.5.1 we show the impact of the flow duration distribution on link dimensioning. Both the importance of the packet correction factor on the estimation of required capacity at smaller timescales and how the packet-level parameters can be fitted are shown in Section 5.5.2. In Section 5.5.3 we show the persistence of fitted packet-level parameters for long term use on link dimensioning, and in Section 5.5.4 we demonstrate the consequences of fitting such parameters with non-Gaussian packet traces. In Section 5.5.5 we show results of the extensive validation of the proposed procedure using the entire measurements dataset. Finally, in Section 5.5.6 we check the accuracy of the proposed procedure in this chapter by means of quantifying the overestimation of obtained estimations.

5.5.1 Choice of Flow Duration Distribution

As explained in Section 5.2, to calculate the flow-level traffic variance one may choose a formula according to the distribution of flow duration. From real flow measurements we have observed that the duration of flow records tend to follow a long-tailed distribution, hence, justifying the selection for a, *e.g.,* Pareto-based variance formula. However, the authors in [112] also show that flow duration do not play an important role in the final estimation of required capacity in the variance. Therefore, difference between estimations using different variance formulas should be negligible. Nonetheless, since in this work we use flow records, which implies a upper-bound for duration, and also by the fact that we classify flow records according to their properties, it is important to revalidate the importance of the flow duration on variance formulas.

Figure 5.3 compares the estimation of required capacity computed using exponential- (Equation (5.2)) and constant-based (Equation (5.3)) variance formulas at various timescales T. These estimations are represented by C_{exp} and C_{const}, respectively. It also plots the estimation curve of empirical capacity C_{emp} to illustrate the cases in which the flow-based estimation is successful. This example shows that the difference between results from both formulas is indeed insignificant and that we can use the simpler constant-based model. Note that in this example we do not implement the packet correction factor ϕ. That is, the flow-based procedure solely gives us a baseline estimation that suffices required capacity at larger T. The packet-level correction factor is, therefore,



(a) Trace from C with $\rho = 0.04$ Gb/s

(b) Trace from D with $\rho = 1.45$ Gb/s

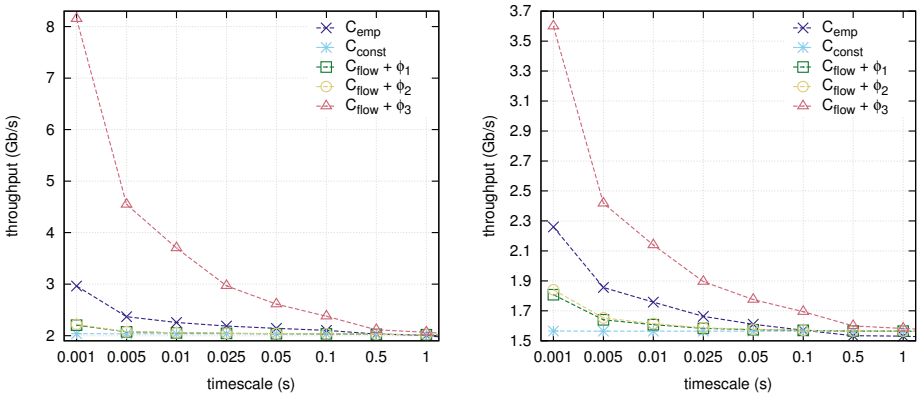
Figure 5.3: Estimation of required capacity at various T; C_{exp} calculated with v_{exp} from Equation (5.2); and C_{const} calculated with v_{const} from Equation (5.3).

needed so that the increasing demand as observed for C_{emp} at smaller T is met. The packet correction factor is validated in the following sections.

5.5.2 Packet Correction Factor

The packet correction factor ϕ helps us to capture packet-level details within flows, ultimately, aiming at better estimations of required capacity at small timescales. Figure 5.4 provides an example of estimation of required capacity C_{flow} using the flow-based model and each one of the three packet correction factors from Section 5.2.3. In this example, all parameters for the packet correction factor formulas were computed out of the measurements. In Figure 5.4, C_{const} is computed using Equation (1.2) with variance $v_{const}(T)$ from Equation (5.3). For C_{flow}, however, the traffic variance is calculated using Equation (5.4) where $v_{flow}(T)$ comes from Equation (5.3).

The packet-level correction ϕ_1, from [112], assumes Poisson packet arrivals and deterministic packet sizes within the flow records. Although better than the purely flow-based method, as shown in Figure 5.4, ϕ_1 is clearly still too optimistic and leads to an underestimation of the required link capacity mainly at small timescales. In ϕ_2 we take into account the influence of the packet size distributions appearing in the formula of Equation (5.6) through the ratio of its second and first moments. The measured values of the first two moments of packet size distribution slightly increases the estimated required capacity, but



(a) Trace from B with $\rho = 1.58$ Gb/s (b) Trace from D with $\rho = 1.45$ Gb/s

Figure 5.4: Estimation of required capacity using different packet correction factors.

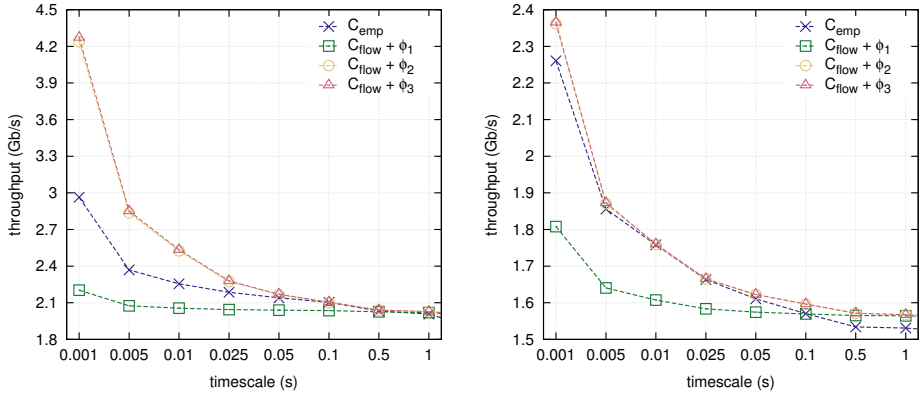
still leads to an underestimation. The main take away of this analysis is that the Poisson packet arrival process within flows is apparently too "friendly". Therefore, in ϕ_3, in addition to the packet size, we explicitly take into account the burstiness of the packet arrival process. This is done by the assumption that the packets arrive according to a compound Poisson process with geometrically distributed batch sizes and then fit (the first and second order statistics of) this process to measurements on the real arrival process. The assumption of a compound Poisson packet arrival process is, however, very conservative (*i.e.*, "too bursty"), which explains the (strong) overestimation of the required bandwidth by ϕ_3, as observed in Figure 5.4.

Since parameters for ϕ_2 and ϕ_3 computed from traffic measurements were not sufficient to provide an accurate estimation of required capacity, we propose such values to be fitted against empirically observed data. It is valid to observe that the fitting procedure does not substitute the model because neither χ nor p depend on other important parameters such as T and ε. Considering how the flow model and the packet correction factor were built, the fitting of a single value of χ or p is done for a specific ε and for any T. Therefore, only one "universal" value of χ or p is obtained for the given trace.

Fitting procedure

The amount of traffic $A(T)$, obtained from packet-level measurements, allows us to compute the ground truth $C_{emp}(T, \varepsilon)$ (see Equation (3.11)). A value for χ or

p is chosen such that the resulting estimation of required capacity C_{flow} satisfies the condition $\hat{\varepsilon} \leq \varepsilon'$ at any T. ε' is the acceptable exceedance probability for the fitting procedure only, *i.e.,* the stopping condition for fitting. The value of ε' should be chosen at most equal to ε so that the fitted values of χ and p would ultimately yield $C_{flow} \geq C_{emp}$ for all considered T.



(a) Trace from B with $\rho = 1.58$ Gb/s

(b) Trace from D with $\rho = 1.45$ Gb/s

Figure 5.5: Estimation of required capacity using packet correction factor ϕ with fitted values of χ and p.

Figure 5.5 shows the estimation curves from the flow-based procedure, using fitted χ for ϕ_2 and fitted p for ϕ_3. The main take away of this figure is that results from the flow-based procedure supported by the packet correction factor are accurate with fitted χ or p since there is no underestimation. However, such accuracy is questionable at T where excessively overestimation happens, *e.g.,* from 1ms to 10ms for Figure 5.5b. Such overestimation happens in situations where $\chi_{T=100ms} > \chi_{T=1ms}$. That is, the estimation of the required capacity requires greater χ at larger T than at shorter ones. We have also observed for the example traces of Figure 5.5 that the packet level correction is needless at $T > 500$ ms. That is, at such timescales $\chi = 0$ and $p = 1$ cancel out the packet correction factors ϕ_2 and ϕ_3, respectively.

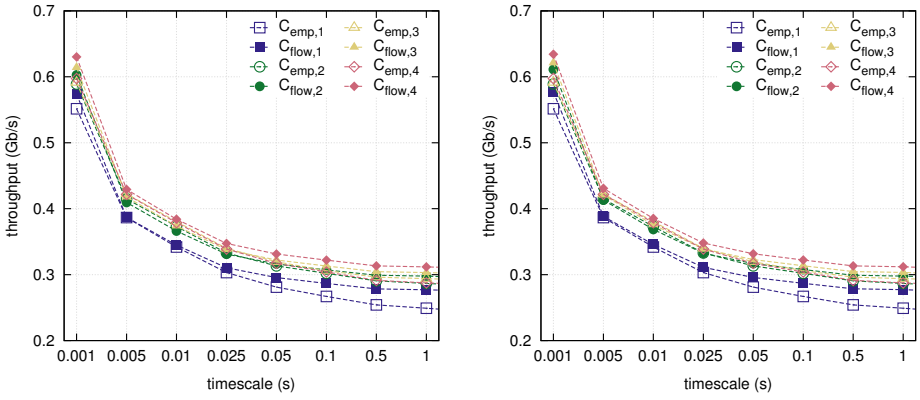
Operators might be interested in a single T or a reduced set of T. In such cases, the fitting procedure can be performed to those specific T only. This would both reduce the execution time of the fitting algorithm and increase the accuracy of the fitted χ or p. The later would help to avoid situations as shown in Figure 5.4b, where the required χ and p differs too much for small and large

values of T. In this case, fitted χ for large T is too high, or p is too low and, hence, they are not an optimal value for the whole range of T.

Now the question is whether a fitted χ or p will remain valid for further successive estimations of required capacity for the same link. Since the fitting process involves packet-level measurements, it is important to minimize such cost as much as possible. That is, if the fitted χ or p can be reused for a long period of time, one will hardly ever need to perform packet measurements for the fitting procedure. The persistence of fitted χ and p is presented in the next section.

5.5.3 Persistence of Fitted χ and p

In the previous section we have shown that fitting χ or p provide us better results at any timescale. However, the drawback is that the fitting process requires packet-level traffic captures to compare the flow-based estimation against an empirical one. The ideal situation would be that the fitted values for χ or p remain valid for a long period of time, providing accurate estimations of required capacity. In this section we show the consistency of fitting χ or p for successive estimations of required bandwidth for the same location. The results in this section used flow records classification by rate and duration and $\varepsilon = 1\%$.



(a) ϕ_2 with fitted χ (b) ϕ_3 with fitted p

Figure 5.6: Estimation of required capacity for four successive traces from location D with χ and p fitted only for trace 1.

Figure 5.6 shows the estimation curves for T ranging from 1ms to 1s. In this figure, the estimations of required capacity for four successive traces from

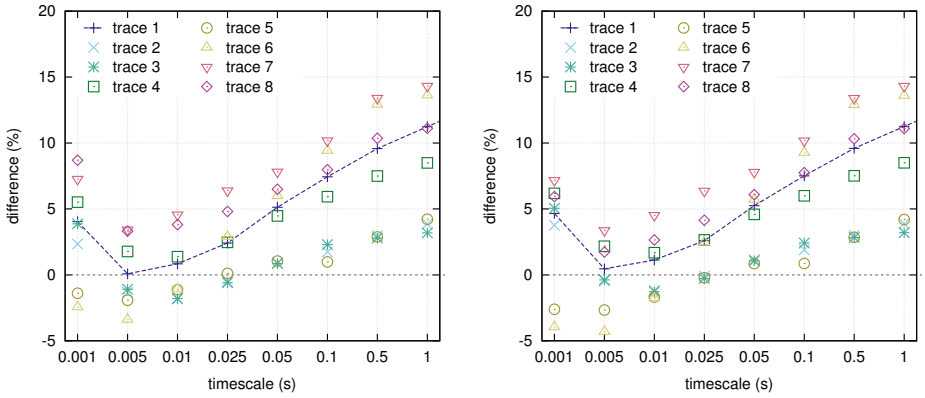
location D are depicted. Figure 5.6a shows the estimations with fitted χ, and Figure 5.6b with fitted p. For both cases, the fitting procedure was performed only for trace 1 of the four traces and the fitted values reused for successive estimations of required capacity. For each trace, the estimation C_{flow} is compared to the trace's respective empirical estimation C_{emp}. The main take away of Figure 5.6 is that C_{flow} is never significantly below the respective empirical C_{emp}. This means that fitted values of χ and p for the first trace were successfully reapplied in further successive estimations for traces from location D.

To extend the example illustrated in Figure 5.6, we assessed the validity of fitted χ and p for a larger sequence of traces from locations D and E. Figure 5.7 shows the relative difference between C_{emp} and C_{flow} for eight traces from location D with fitted χ or p. Figure 5.8 shows the same results for eight 15-minute traces from location E. In both figures, the first 4 traces (traces 1 to 4) were captured roughly two months before the last four traces (traces 5 to 8). Notice that the eight 15-minute traces from location D in Figure 5.7a are the same as the ones in Figure 5.7b. This remark also applies to the eight 15-minute traces from location E used in plots of Figure 5.8.

The plots in Figures 5.7 and 5.8 show the difference in percentage of the calculated C_{flow} using ϕ_2 or ϕ_3 and C_{emp}. That is, y-axis represent how much the obtained C_{flow}, using fitted χ from trace 1, underestimates or overestimates the empirical required capacity C_{emp} at different T. Clearly, due to the fitting procedure, for trace 1 $|C_{flow} - C_{emp}| \geq 0$ (*i.e.,* no underestimation). However, one can see that the overestimation at short T is not very high, most traces are below 10% for any T. It means that the obtained exceedance probability $\hat{\varepsilon}$ for such cases is less than, but also close to the defined 1% for ε. There are also cases of underestimation, but these are not less than -5%. This means that the obtained error $\hat{\varepsilon}$ is not much higher than the defined ε.

Note that values for fitted χ and p are very similar in all cases of Figures 5.7 and 5.8. However, few differences can be observed. For example, in Figure 5.8, at $T = 1$ms, the approach using ϕ_2 resulted in higher underestimation for traces 5 and 6 than the approach using ϕ_3.

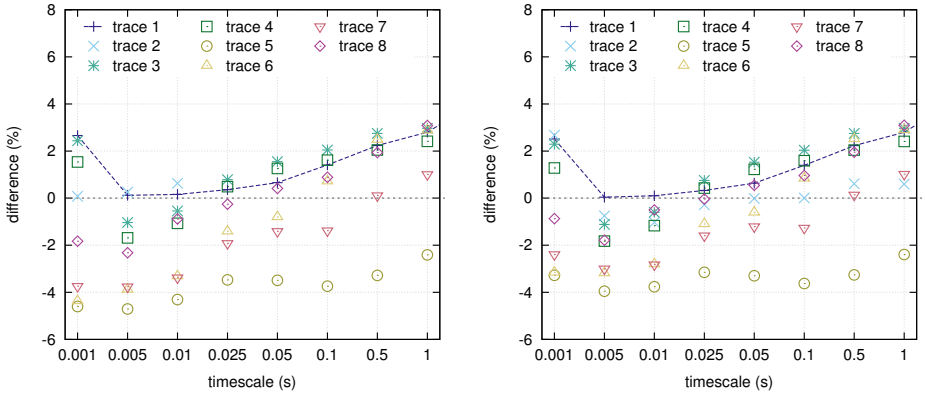
The main conclusion of Figures 5.7 and 5.8 is that the fitted value of χ or p for a single trace remained valid for several successive traces, supporting accurate estimation of required capacity and keeping differences between estimations very small, specially at shorter T. The fitting procedure inherits from the dimensioning formula of Equation (1.2) the dependency on Gaussian traffic. Therefore, fitting with non-Gaussian traffic may not yield expected results. This problem is better detailed in the next section.



(a) ϕ_2 with fitted χ for the trace 1

(b) ϕ_3 with fitted p for the trace 1

Figure 5.7: Relative difference between C_{flow} and C_{emp} for eight successive traces from location D with χ and p fitted only for trace 1.



(a) ϕ_2 with fitted χ for the trace 1

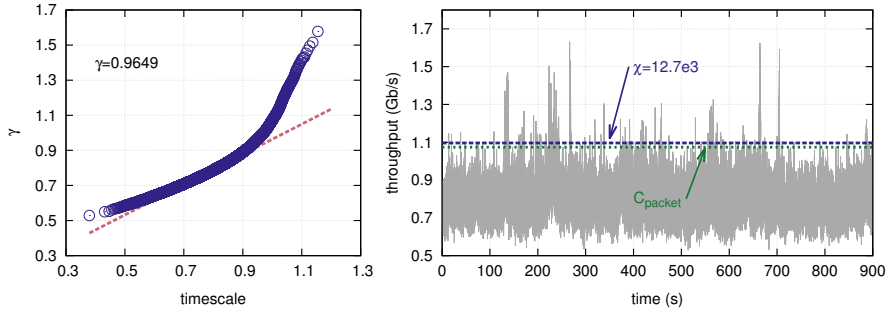
(b) ϕ_3 with fitted p for the trace 1

Figure 5.8: Relative difference between C_{flow} and C_{emp} for eight successive traces from location E with χ and p fitted only for trace 1.

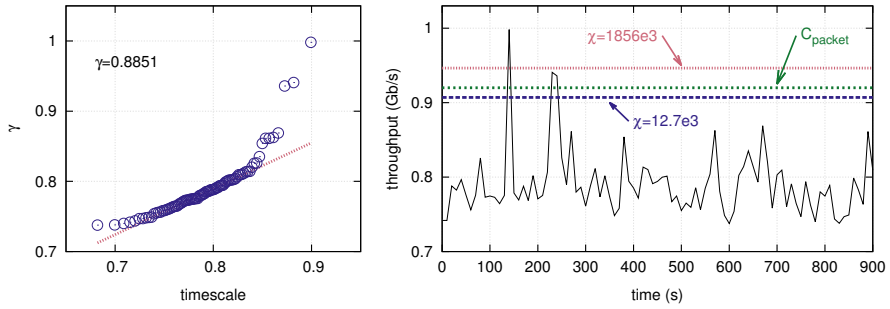
5.5.4 Fitting with non-Gaussian Traces

One of the key requirements of the link dimensioning formula of Equation (1.2) is that input traffic is Gaussian (*i.e.,* normal-distributed). Obviously, such requirement also extends to the fitting procedure, since the dimensioning formula

is used. Attempting to fit χ or p using non-Gaussian traffic might result in unexpected behavior of the fitting procedure. In this section we use an example trace from location E that is non-Gaussian at larger timescales. Although it is expected that traffic is presumably less Gaussian at smaller T [71, 110], we have demonstrated in Section 2.4.3 that traffic might also loses its Gaussian properties at larger T [40, 41].



(a) $T = 10$ms



(b) $T = 10$s

Figure 5.9: Time series and estimations of required capacity using fitted χ for example trace from location E at different timescales.

In the example trace used in this section, several traffic bursts of millisecond-precision occurred close to each other in time, as one can see in the time series of Figure 5.9a. Those bursts, however, did not impact negatively on Gaussian fit, since at $T = 10$ms this trace had $\gamma = 0.9649$, as seen in the Q-Q plot of the same figure. By increasing the size of the bins in the time series, *i.e.,* $T = 10$s, the nearby bursts were averaged within the same time bin, as shown in Figure 5.9b. This resulted in long-lasting traffic bursts with rates much

higher than the average trace rate at such timescale. These long-lasting peaks compromised the Gaussianity fit of the trace at $T = 10$s.

When executing the fitting procedure with the example trace of Figure 5.9, the yield values for χ and p do not make sense at larger timescales. The proposed packet correction factor is intended for helping the flow-based model to estimate required capacity at shorter T. Therefore, the larger the T, the more we expect that $\chi \sim 0$ and $p \sim 1$. That is, the packet correction factor is cancelled since it is not needed at larger T (see Figure 5.4 and 5.5). However, in this example trace, the fitted values of χ and p are more conservative at large T (*i.e.,* estimations of required capacity are much higher than actually needed).

Consider that the stop condition for the fitting procedure is set to $\varepsilon' = 1\%$. The time series of Figure 5.9a consists of 10k bins of size $T = 10$ms. This means that the fitting procedure allows for 900 of these bins to have values above the estimated $C_{flow}(T, \varepsilon')$. Under these parameters, we obtain $\chi = 12700$. When defining $T = 10$s, nearby traffic peaks between 100s and 300s of the time series in Figure 5.9b are averaged within few time bins, resulting in huge bursts. At $T = 10$s, the definition $\varepsilon' = 1\%$ allows for only 0.9 bins to have traffic rate above the estimated $C_{flow}(T, \varepsilon')$. This results in $\chi > 5 \times 10^6$. Even if we consider the interpolated value between the 99-th and 100-th percentiles of the trace traffic rate distribution, the fitting procedure unreasonably yields $\chi > 1.8 \times 10^6$.

The main take away of this analysis is that the resulting traffic peaks at $T = 10$s demand disproportionally high values of χ so that the fitting condition of $\varepsilon' = 1\%$ is met. Considering practical deployment of link dimensioning, $\chi_{T=10s} = 12700$ would suffice, since the operator would be interested in finding a long lasting χ that potentially takes care of regular traffic bursts and disregards unusual peaks – *i.e.,* focusing on customary network behavior and not on exceptions.

It is important to mention that at higher timescales the packet-based link dimensioning approach from [109] also fails to estimate the required capacity of traces that behave in a similar way than the example trace in this section. In the next section we present results of a thorough validation of the proposed hybrid flow-based approach using all traces from the dataset introduced in Chapter 2.

5.5.5 Extensive Validation and Overall Results

In this section we validate the proposed flow-based procedure for link dimensioning by estimating the required capacity for all traces in our dataset. We present results for both packet correction factors ϕ_2, from Equation (5.6), and ϕ_3, from Equation (5.8). A single fitting of χ and p is done for each location using the very first trace in chronological order. Then, the obtained values for χ and p are reapplied to all successive traces for each location. Our conclusions on the

quality of estimations are drawn based on the obtained exceedance probability $\hat{\varepsilon}$ given by Equation (3.12). As well as in previous sections, we used *a60i20* flows, *i.e.,* created with active timeout of 60s and inactive timeout of 20s. Resulting flow records were classified by their respective rate and duration, following the procedure detailed in Section 5.3. The parameters used for the classification were $\theta = 1000$ bytes/s and $\eta = 100$ms. To comply with previous works, the exceedance probability was set to $\varepsilon = 1\%$ and T varied from 1ms to 1s.

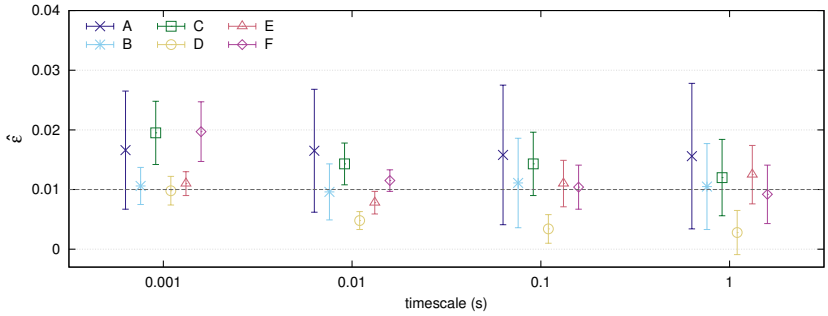
> ***Recap:*** *The estimation of required capacity is considered successful when it yields $C(T, \varepsilon) \geq C_{emp}(T, \varepsilon)$, which is by definition the same as $\hat{\varepsilon} \leq \varepsilon$.*

The summary of results is shown in Figure 5.10, where for each location the average and standard deviation of $\hat{\varepsilon}$ at various T are plotted. One can see the small difference on results between the approach using ϕ_2, in Figure 5.10b, or the one using ϕ_3, Figure 5.10c. Nonetheless, the approach using ϕ_3 is slightly more conservative. In addition, for comparison purposes, Figure 5.10a shows the average and standard deviation of $\hat{\varepsilon}$ for the purely packet-based approach as proposed in [109]. That is, mean traffic rate and variance were calculated directly from packets. The main take away of this comparison is that our proposed approaches, helped by the packet correction factors, manage to achieve more conservative estimations at short T, but they demonstrate to be more unstable at large T. The purely packet-based approach was successful (*i.e.,* $\hat{\varepsilon} \leq \varepsilon$) in about 22% of all traces at $T = 10$ms and 48% at $T = 1$s. Due to conservative estimations, our procedure using ϕ_2 correctly estimated required capacity for 64% of traces at $T = 10$ms and for 22% at $T = 1$s. Furthermore, using ϕ_3, success was improved to 87% of traces at $T = 10$ms and 28% at $T = 1$s.

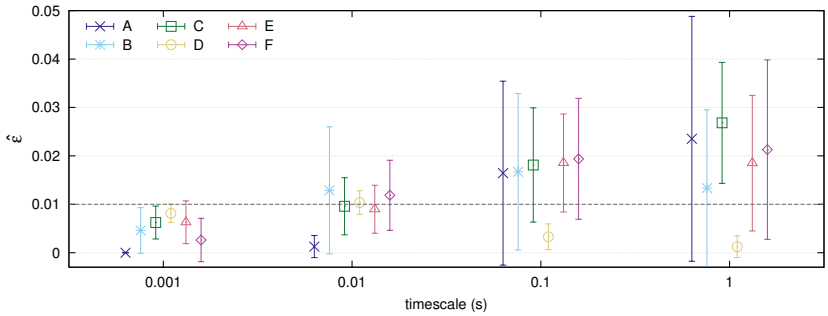
For the proposed procedure, the worse estimations at $T \geq 100$ms can be related to fitting of parameters χ and p using non-Gaussian traces. As explained in Section 5.5.4 this would result in better estimations of required capacity for shorter T, but traffic bursts at larger T would result in higher $\hat{\varepsilon}$. Therefore, underestimation problems at larger T could be alleviated by assuring χ and p to be fitted using Gaussian traffic. Nonetheless, if one considers all estimations of required capacity that resulted in a not too high $\hat{\varepsilon}$, let's say less than 2%, results become more expressive for our procedure. In such case, for the proposed procedure using ϕ_2, 95% and 44% of traces had $\hat{\varepsilon} \leq 2\%$ at $T = 10$ms and $T = 1$s, respectively. For our procedure using ϕ_3, 99% and 59% of traces had $\hat{\varepsilon} \leq 2\%$ at $T = 10$ms and $T = 1$s, respectively.

5.5.6 Quantifying the Overestimation

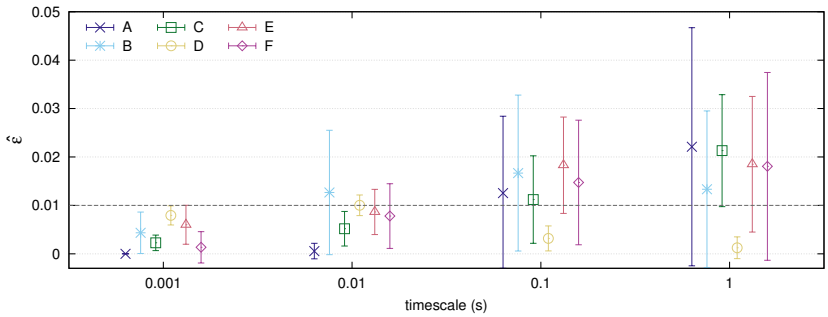
Although $\hat{\varepsilon} \leq \varepsilon$ is desirable, excessive overestimation is not. If overestimation happens it should be between reasonable boundaries, *i.e.,* not overly higher



(a) packet-based link dimensioning



(b) flow-based link dimensioning using ϕ_2 with fitted χ



(c) flow-based link dimensioning using ϕ_3 with fitted p

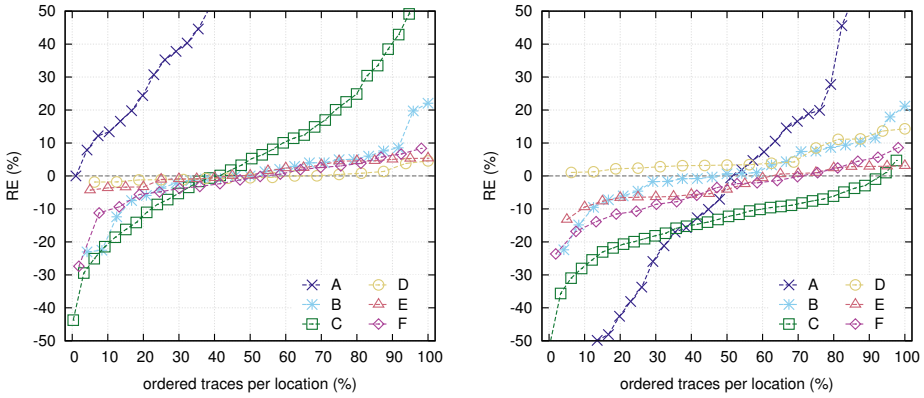
Figure 5.10: Average and standard deviation of $\hat{\varepsilon}$ per location for all traces in our dataset.

than the empirical capacity C_{emp} for any T and ε. For example, in the plots of Figure 5.10b and 5.10c one can see that at very small timescales $\hat{\varepsilon} = 0$ for location A and the standard deviation is insignificant. The reason for this becomes clear when computing the relative error RE, from Equation (3.13). Figures 5.11 and 5.12 show the normalized RE for all traces in our dataset. Note that, since there are different number of traces per location, the x-axis in the plots of these figures shows the percentage of traces per location sorted from left to right by their respective RE.

> ***Recap:*** *The relative error RE is used to assess how much for more (overestimation) or for less (underestimation) the estimated required capacity $C(T, \varepsilon)$ differs from the empirical ground truth $C_{emp}(T, \varepsilon)$.*

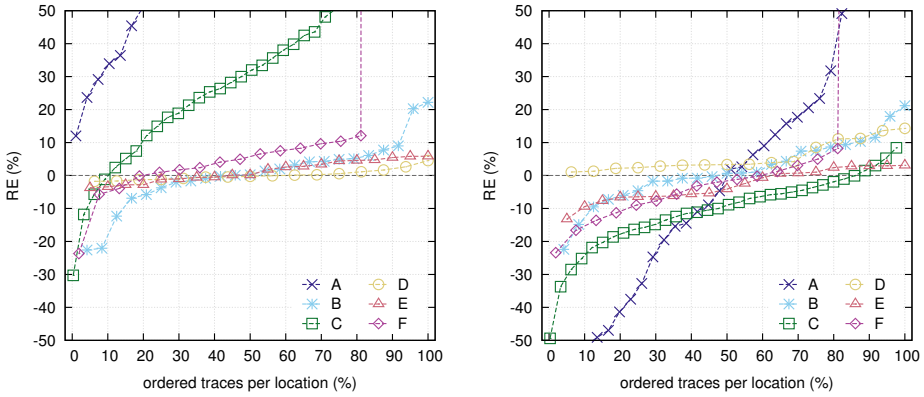
In Figures 5.11 and 5.12 one can see that the overestimation is actually quite high for most traces from A at small T. Using ϕ_3 with fitted p at $T = 10$ms (Figure 5.12a), for only about 15% of traces from A the $C(T, \varepsilon)$ is less than 50% more the $C_{emp}(T, \varepsilon)$. This problem, although with less intensity, can also be observed for traces from location C. As previously mentioned, in our experiments we fitted χ and p only once. Locations A and C illustrate what happens when the shape of the traffic in the measured link constantly varies. Since the measured link in these locations carries traffic of a small number of users, it only takes few users to change traffic properties and invalidate previously fitted χ and p. Besides, these measurements also capture differences in traffic due to day and night patterns. By fitting parameters only once, the "bad fitting" was never fixed and for the other remaining traces the fitted values of χ and p were not the correct ones and, ultimately, yielded mostly very conservative results. For such networks, a system implementing the proposed link dimensioning procedure would better to also implement a checking process to, *e.g.,* decide whether to run the fitting of parameters again once exorbitant values of estimated required capacity were obtained (*i.e.,* the fitting process should be performed again aiming at having proper values for χ or p). Another idea would be to use different values of χ and p fitted at different times of the day.

Considering only traces from C, at $T = 1$s and using ϕ_2 with single fitting of χ in around 76% of traces the estimated $C(T, \varepsilon)$ was kept in between 20% for more or less the empirical estimation $C_{emp}(T, \varepsilon)$. At the same timescale and using ϕ_3 with fitted p, around 84% of traces had estimated capacity within this range. For most of the traces of the other locations (with a larger and regular number of active users throughout the measured period) the estimated required capacity $C(T, \varepsilon)$ remained between reasonable bounds, *i.e.,* between 20% for more or less the empirical estimation. For example, at $T = 10$ms, excluding those from locations A and C, using ϕ_2 around 96% of all traces had estimated



(a) $T = 10$ms (b) $T = 1$s

Figure 5.11: Relative Error for all traces per location using ϕ_2 with fitted χ; y-axis is limited to $[-50..50]$ for visualization reasons.



(a) $T = 10$ms (b) $T = 1$s

Figure 5.12: Relative Error for all traces per location using ϕ_3 with fitted p; y-axis is limited to $[-50..50]$ for visualization reasons.

$C(T, \varepsilon)$ within the range of 20% for more or less the empirical estimation and, using ϕ_3, it was more than 87% of all traces. In the latter case, for few and not necessarily consecutive traces from F, the fitted value of p was not appropriate, leading to excessive overestimation of required capacity.

5.6 Operational Considerations and Selection of Parameters

The proposed flow-based procedure relies in a number of parameters. This section is dedicated to discuss the parameters that were not presented in previous sections and their respective impacts on the accuracy of the proposed link dimensioning procedure.

5.6.1 Measurement duration

In this paper we have only used 15-minute long traces, hence, simulating traffic being monitored every 15 minutes. The measurement duration should be reasonably chosen such that the traffic during the measurement can be considered stationary, as required by the dimensioning formula of Equation (1.2). Longer periods might capture undesired periodic changes on traffic behavior hurting its stationarity character. However, that's not true to assume that traffic will always be stationarity when measured in periods of 15 minutes. It will depend on the traffic nature and network users behavior. The measurement period of 15 minutes used in this paper was chosen to comply with previous work [109].

5.6.2 Flow timeouts

The active timeout t_a and inactive timeout t_i are set on the flow exporter and they define the length of a flow record and, consequently, the level of aggregation of traffic information. The chosen timeouts will depend on the purposes of traffic monitoring at the network operator. The analysis of previous sections were presented using flow records created with $t_a = 60$s and $t_i = 20$s. However, we have tested our proposed flow-based link dimensioning procedure using many other combinations of timeouts, varying t_a from 5s to 120s and t_i from 2s to 30s (always obeying the condition $t_a > t_i$). We have not observed any significant difference between results obtained with different timeouts and, therefore, we assume that, for the tested range of values, the timeouts combination does not impact on the accuracy of the estimated required capacity. It is important to know, however, that the amount of processed flow records is the most dominating factor in the computation time in the proposed flow-based procedure.

5.6.3 Flow records classification

In the previous sections, we presented results obtained with flow records classified by rate $\theta = 1000$ bytes/s and duration $\eta = 100$ ms. Since the definition of these parameters depends on the traffic nature, the network operator would

also be responsible for such task. By testing the proposed flow-based procedure we have observed that the smaller the θ and η parameters are defined, the more accurate is the estimation of required capacity. However, the smaller are such parameters the more classes will be created and, consequently, the more time the procedure may take to compute the required capacity. From the results presented above we can conclude that the settings used in this paper are enough for providing satisfactory accuracy on estimations of required capacity. It should be emphasized that the proposed link dimensioning procedure is very lightweight and even a standard computer can perform the computations for 20K flow classes in few seconds.

5.6.4 Exceedance probability ε

To comply with previous works [96, 109, 112], in this paper we have always set $\varepsilon = 1\%$. Clearly, it does not make sense to choose smaller ε at large T when the measurement duration is no longer than 15 minutes (as in the case of this paper). For example, setting $\varepsilon = 1\%$ at $T = 10$s means that the dimensioning formula should return an estimated required capacity so that under-provisioning happens in only 0.9 out of 90 time bins. Consequently, the link dimensioning procedure may result in excessive overestimation so that over-provisioning happens for all time bins. In addition, network operators must take into consideration the length of the time bin defined by T. That's because the larger T the more traffic is aggregated within a single time bin. This means that, depending on the link load, a single under-provisioned time bin at $T = 1$s might result in much bigger problems of performance than a under-provisioned time bin at $T = 10ms$. Therefore, ε must be chosen to avoid underestimation but also avoiding unnecessary overestimation.

5.6.5 Fitting of χ and p

The crucial point of the fitting procedure for the packet corrections ϕ_2 and ϕ_3 is that ε' should be chosen such that $C_{flow} \geq C_{emp}$. The chosen value for ε' should be enough to avoid underestimation but also not too conservative so that overestimation is not excessively high. For example, $\varepsilon'_1 = \varepsilon$ will result in $C_{flow,1} = C_{emp,1}$ and $\hat{\varepsilon} = \varepsilon$. If the fitted value of χ or p is subsequently used for estimating required bandwidth of the next 15-minute measurement period, and $C_{flow,2} < C_{emp,2}$, the end result may be the undesired $\hat{\varepsilon}_2 > \varepsilon$. Therefore, ε' should be wisely chosen obeying $\varepsilon'_2 < \varepsilon$. This way $C_{flow,2} > C_{emp,2}$ for the fitted trace, and a safety margin is kept in order to assure $\hat{\varepsilon}_2 \leq \varepsilon$ for successive traces using the previously fitted χ or p. To play safe, the network operator may choose $\varepsilon' = \varepsilon$, as done in the experiments in this paper. To further reduce risks

of having many underestimation cases for successive traces, χ and p should be fitted only using Gaussian traces.

5.6.6 Choice between ϕ_2 and ϕ_3

Concerning the packet correction factors ϕ_2 and ϕ_3, we have showed that the latter provided better results than the former. However, this small gain comes at a cost. The trade-off is that ϕ_3 requires the second moment of packets size $\mathbb{E}[S^2]$ (see Equation (5.8)). A simple modification in the flow exporter is needed so that the sum of squares of packets size is also exported within the flow record. Therefore, ϕ_3 deployment is limited to cases in which network operator is able and willing to modify the flow exporter. Modifications can easily be made if the operator uses an open-source flow monitoring tool. For example, we have implemented such modifications in the open source exporter YAF [61].

5.6.7 Procedure Execution Performance

Regarding the performance of the whole link dimensioning procedure, one can divide it in two parts: the traffic measurements and the dimensioning calculations. Although our proposed procedure requires sporadic packet-level traffic measurements for the fitting of parameters, these captures do not need to happen for long periods and the main basis of the procedure relies solely on flow-level measurements. It remains, therefore, a lightweight procedure in terms of traffic measurements. Concerning execution time of the calculations for estimating the required capacity, we have observed that even for the largest traces (*i.e.,* those from location D, E and F) the whole procedure usually took less than a minute to complete. For example, for a large 15-minute trace from location D, our procedure classified more than 5.6 million flows (defined as *a60i20*) by their respective rate and duration into almost 14000 classes (defined by $\theta = 1000$ bytes/s and $\eta = 100$ms, which is the most granular classification tested by us). For each class a variance was computed using Equation (5.4). These variances were summed up and, with the trace average rate ρ, applied into the link dimensioning formula. Ultimately, the estimation of required capacity $C(T, \varepsilon)$ was obtained for the same range of timescales used throughout this paper (*i.e.,* 1ms to 1s). The overall procedure took around 50 seconds to complete. Nonetheless, the most costly operation in the whole proposed procedure is the fitting process of parameters for the packet correction factor. This process mainly depends on the range of timescales of interest. The larger the range, the longer the fitting process takes to fit the parameters such that the condition $\hat{\varepsilon} \leq \varepsilon$ is satisfied for all the considered timescales. Using the same example trace from D, it took around 1min45s for fitting p (used in ϕ_3) for the same range of timescales from

1ms to 1s. Note that these time measurements come from a prototype brute-force implementation (mix of C and shell scripts). System performance was not the focus of this paper. However, one can certainly expect significantly lower run times with a proper production-ready implementation.

5.6.8 Link dimensioning in practice

It is inevitable that network operators, even having good estimations of required capacity for their links, will eventually add safety margins on the top of these estimations. As mentioned in the beginning of this paper, this is already adopted in practice. However, nowadays operators add margins on top of traffic averages obtained from reading SNMP counters at very coarse time resolutions, such as 5-minute averages. The procedure proposed in this work comes to add more reliability on link dimensioning by providing a well founded baseline estimation. Independently of adding or not a safety margin on top of the estimations, our procedure proved to be, at finer time resolutions, as much efficient as a packet-based approach. Nonetheless, by adding a safety margin, problems of underestimation of required capacity due to, *e.g.,* fitting of parameters with non-Gaussian traces, can be alleviated. For instance, in cases of Figures 5.7 and 5.8 around 5% extra capacity (*i.e.,* on top of the estimated one) would already be sufficient for all considered traces to have $\hat{\varepsilon} \leq 1\%$.

5.7 Concluding Remarks

In this chapter we proposed a hybrid flow-based link dimensioning approach. Our approach extends the work from [112] by adding a method to capture packet-level details besides the flow-level ones. The proposed approach in this chapter provides a well founded baseline estimation of required capacity for network traffic streams. By using traffic measurements at the flow level, and seldom requiring packet captures, our proposed procedure is – almost – as easy-to-deploy as SNMP-based approaches and with the advantage that it allows to gather information about traffic fluctuations at finer time resolutions, which was the main drawback of the pure flow-based approach propose in Chapter 4.

The main advantage of our procedure is that by integrating analytical modeling with measurement data, estimations of required capacity are – almost – as accurate as fully packet-based approaches without the overhead of performing continuous packet captures. Although requiring packet-level measurements, we have demonstrated that the fitted values for χ and p using 15-minute long packet captures remained valid for very long periods of time. Measurement-wise, this is a lightweight approach compared to the fully packet-based approach from [109].

Moreover, the proposed approach in this chapter is able to estimate the required bandwidth within seconds even when several thousands of flows are measured.

Our findings showed that our procedure is able to accurately estimate the required capacity for a range of time resolutions as low as 1 millisecond. For most applications, *e.g.,* web browsing, end users usually experience QoS at the timescale of 1 second. However, having accurate estimations of required capacity for shorter timescales can help ensuring QoS for delay sensitive applications, such as VoIP and real-time video streaming.

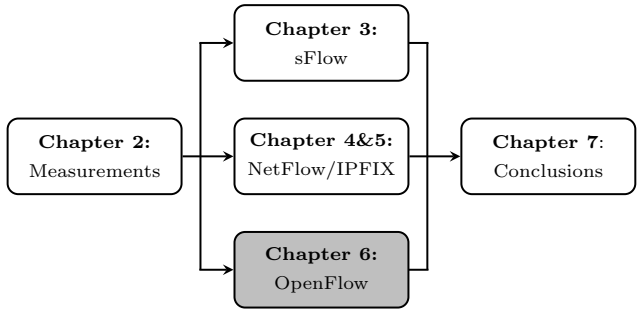
I, a universe of atoms, an atom in the universe.

— Richard P. Feynman

CHAPTER 6

OpenFlow-based Link Dimensioning

Software-defined Networking (SDN) has recently gained lots of attention from the research community and industry. The main idea behind SDN is the decoupling of the control and data planes, which allows for the underlying infrastructure to be abstracted for network applications and services. OpenFlow is a largely adopted standard to implement SDN. Besides enabling the communication between control and data planes, OpenFlow also implements basic traffic monitoring and measurement operations. In this chapter we introduce an approach to use OpenFlow traffic measurements for link dimensioning. We also present a preliminary study on the quality of traffic measurements obtained from current implementations of the OpenFlow protocol. The publication related to this chapter is [37].



The organization of this chapter is as follows:

- *Section 6.1 motivates the research presented in this chapter and states our contribution.*
- *Section 6.2 introduces the OpenFlow standard focusing on the maintenance and monitoring of traffic data.*
- *Section 6.3 proposes an approach to retrieving traffic measurements from OpenFlow for link dimensioning purposes.*
- *Section 6.4 assesses the quality of OpenFlow traffic measurements.*
- *Section 6.5 concludes this chapter.*

6.1 Background

Recently, SDN has gained lots of attention from the research community due to its – although questionable to some – revolutionary idea of separating the control and data planes on the network. The focus is on abstracting the underlying infrastructure from the application and services, which may simplify network management. For example, the authors in [103] demonstrated that it was possible to simplify the implementation of MPLS VPNs using OpenFlow.

OpenFlow is certainly the most famous technology within the context of SDN. OpenFlow is a specification that includes switch, controller and protocol components. The current version of OpenFlow specification is 1.4 [91]. However, most available OpenFlow implementations are based on the 1.0 version [89]. Many network vendors already include OpenFlow within their switch and router devices. Among the many functionalities of the OpenFlow switch, it is of our interest that the OpenFlow switch is responsible for performing some basic traffic measurements. The OpenFlow switch meters statistics of observed flows by means of counters (bytes and packets). The value of these counters can be retrieved by the OpenFlow controller (i) at any time in a push fashion by requesting such information to the switch, or (ii) occasionally when receiving some specific messages from the switch – in Section 6.2 we detail these operations. The metered statistics can be used by applications running on top of the controller.

To the best of our knowledge, the use of OpenFlow traffic measurements for link dimensioning has not yet been explored. Nonetheless, several works have used traffic measurements from OpenFlow to support other operations such as traffic engineering [108, 113, 115] and security [118]. However, traffic measurements in SDN might be unreliable due to many reasons, from hardware limitations to network setup (*e.g.,* limited bandwidth between switch and controller). Many works have addressed this problem in different ways. For example, in [68] the authors aimed at detecting large traffic aggregates by periodically adjusting wildcard rules in the OpenFlow switch and reducing switch overhead. This study later led to [116] where a measurement framework is proposed for SDN networks. In [84, 117] authors explore the relation between hardware resources and measurements accuracy. There are other works proposing different ways to measure traffic in SDN networks. For example, [23] proposes a monitoring framework that actively measures traffic by polling statistics from OpenFlow switches, while in [115] authors propose a passive approach that only uses reporting messages from OpenFlow switch to controller. The quality of pure OpenFlow measurements is assessed in [64]. The drawback of such work is, however, that `Iperf` was used to generate synthetic UDP traffic at a maximum rate of 1 Mbps, which clearly is not representative of real traffic rates.

Finally, [8] surveys traffic engineering solutions for SDN networks, giving special attention to, among others, tools for traffic analysis.

6.1.1 Contribution

Motivated by the increasing interest on SDN and the fast-growing adoption of OpenFlow, in this chapter we introduce an approach to use traffic measurements obtained via OpenFlow protocol for link dimensioning purposes. OpenFlow currently focus on traffic forwarding and management, but not on measurements. Therefore, OpenFlow measurements are still rough when compared to NetFlow. In this chapter we also assessed the quality of traffic measurements provided by current implementations of the OpenFlow protocol and whether the measurement data can be used for link dimensioning. This study is done in both a virtual and real deployment.

In the approach of this chapter we define *OpenFlow flows* in the same way we define *NetFlow/IPFIX flows* in Chapters 4 and 5. By doing so, OpenFlow can theoretically provide us with very similar measurements to those provided by NetFlow/IPFIX technologies. Therefore, it is important to understand that in this chapter we do not develop a new link dimensioning approach, but we introduce a method to obtain flow-level traffic measurements using OpenFlow. The measured data from OpenFlow can be applied to one of the flow-based approaches proposed in Chapters 4 and 5.

6.2 OpenFlow

In this section we provide a brief overview on the operation of OpenFlow. For brevity's sake we only focus on operations that are relevant to the problem of traffic measurement and link dimensioning. One can find a more complete overview of the OpenFlow standard on, what is by the time of this thesis, its latest specification [91].

6.2.1 Overview

OpenFlow was originally proposed in [83] to enable researchers to run experiments in campus networks. By decoupling the control and switch planes, OpenFlow allowed for the separation of research and production traffic. This way, experiments could easily be done with research traffic without impacting on production traffic. Decoupling control and data planes results, on the latter, in a "dumb" switch designated to merely execute tasks without having a saying on, *e.g.,* routing decisions. On the control side, the OpenFlow controller im-

plements all the intelligence needed to, *e.g.,* decide on which port to forward a packet based on one or even multiple routing protocols.

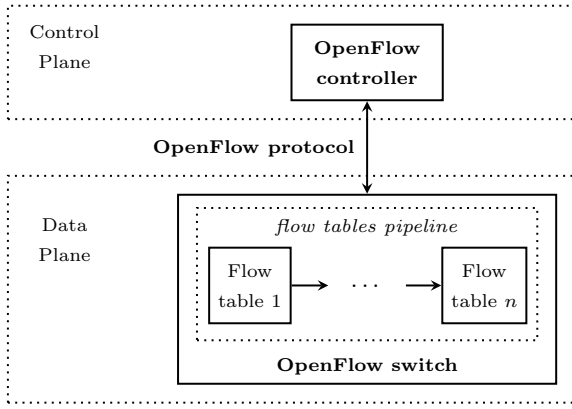


Figure 6.1: Overview of OpenFlow components.

As shown in Figure 6.1, the OpenFlow standard defines three basic components, namely controller, switch and protocol. The OpenFlow protocol enables the communication between controller and switch. In an OpenFlow network the switch can be either an OpenFlow-only or an OpenFlow-hybrid switch. While the former only operates OpenFlow, the latter is a typical Ethernet switch with an embedded OpenFlow implementation. The OpenFlow-hybrid switch must have a *decision mechanism* that decides whether a packet should be treated as ordinary traffic, or processed by the OpenFlow that is running in parallel to the regular switch. The decision mechanism is vendor-specific and it is not part of the OpenFlow specification.

6.2.2 Flow Tables

In the OpenFlow switch a *flow table* contains packet forwarding instructions and per-flow statistics. Forwarding instructions are managed by the OpenFlow controller via messages exchanged with the switch, as detailed in the next section. The switch is solely responsible for maintaining per-flow statistics, such as packet and byte counters. A OpenFlow switch must have at least one flow table, but might have multiple in what is called flow table pipeline. Clearly, the more tables the longer an incoming packet might take to be processed and forwarded. Each entry in a flow table consists of information related to a single flow only. A flow table entry consists of, among others:

- the *match fields* used to identify the flow entry which an incoming packet belongs to;
- the *action set* consisting of what to do with the packets that match the entry's matching fields;
- the *priority* that defines from which entry the switch must execute the set of instructions in case of multiple matches for a single packet;
- *counters* that keep record of basic statistics about the flow;
- *timeouts* that define the entry's lifetime.

The *match fields* for a flow can be various, which makes OpenFlow very flexible on the definition of flows. A flow can be defined as an IP connection, for example, using the 5-tuple key as in NetFlow v5: source and destination IPs, source and destination ports and transport protocol (as described in Section 4.1.2). OpenFlow also allows for a broader definition of flows, *e.g.,* by a VLAN tag or by a port number only. Moreover, two different flows within the same table can have completely distinct match fields, *e.g.,* one defined by the 5-tuple and the other defined by a VLAN tag. The OpenFlow controller is responsible for defining the match fields for a flow table entry. The table flow entry that wildcards all fields (*i.e.,* match any packet) is called the *table-miss flow entry*, and it has the goal of dealing with packets of unknown flows. According to the OpenFlow standard, every table must have a table-miss flow entry.

The *action set* of a flow table entry defines what to do with packets whose properties match the entry's match fields. The action can be the forwarding of the packet to a specific OpenFlow port, or further searching for an action set in another table. In case of a pipeline of multiple flow tables, the walking through tables always goes forward. That is, the set of instructions of an entry in a flow table cannot have an action to move back to a table whose index is lower than the current one. In particular, the action set of the table-miss entry should support sending a message to the OpenFlow controller to inform about the received packet that does not match any flow definition. This situation is further explained in Section 6.2.3.

Every flow table entry is given a *priority*. That's because the flexible and broad definition of flows in OpenFlow allows for situations that a single incoming packet matches multiple table entries. For example, let's consider a flow table entry with priority 1 that has match fields defined by the source IP address, and another flow table entry with priority 0 that is defined by a VLAN tag. If a single packet matches both entries, the OpenFlow switch will execute the action set of the entry with highest priority only, in this case priority 1. The

table-miss flow entry has priority 0, the lowest possible. That is, its action set is only executed if, and only if, the incoming packet does not match any other table entry.

Each flow table entry maintains some basic statistics *counters*. As defined in the OpenFlow specification [91], the switch should keep on a per-flow basis the number of received packets and bytes and the flow duration.

Finally, for every flow table entry two *timeouts* are fixed, namely *hard timeout* and *idle timeout*. Their definition is very similar to the active and inactive timeouts in NetFlow, as explained in Section 4.1.2. The hard timeout defines the maximum lifetime (in seconds) of a flow table entry, and the idle timeout defines how long (in seconds) an entry remains in the flow table after the receiving the last matching packet. When either the hard or the idle timeout of an entry expires, the entry is removed from the flow table and, if previously established, the OpenFlow controller is notified of the removal. Given that tables in the OpenFlow switch must always have a flow-miss entry, the timeouts for the flow-miss entry should be set to zero (*i.e.,* undetermined), preventing it to expire.

6.2.3 Protocol Messages

The communication between OpenFlow controller and switch is enabled by the OpenFlow protocol. It supports three message types. The *controller-to-switch* messages are initiated by the controller with the purpose to manage and check the switch. The *asynchronous* messages are initiated by the switch with the purpose to update the controller about various changes, such as on flow table entries. The *symmetric* messages can be started by either the controller or the switch. Symmetric messages are used for handshake upon the establishment of connection between the controller and the switch, and also for the switch to report problems to the controller.

An exhaustive description of the messages of OpenFlow protocol can be found in the OpenFlow specification [91]. For brevity's sake, however, next we describe only the messages that are relevant to our work, *i.e.,* traffic measurements for link dimensioning purposes. Figure 6.2 helps to understand the exchange of these messages between controller and switch. The OpenFlow messages that we are interested in are the following:

- `OFPT_PACKET_IN`: message sent from the switch to the controller every time a packet does not match any of the flow table entries but the table-miss flow entry. Notice that sending an `OFPT_PACKET_IN` message is not mandatory for the actions set of a table-miss entry, which could simply drop the received packet.

- `OFPT_FLOW_MOD`: message used by the controller to add, modify or delete a flow entry in a specific table at the switch. An important feature is that the flag `OFPFF_SEND_FLOW_REM` should be set in an `OFPT_FLOW_MOD` that is adding or modifying a flow entry. This way once a flow entry is removed by the switch due to timeout expiration, the switch sends an `OFPT_FLOW_REMOVED` message to the controller.

- `OFPT_FLOW_REMOVED`: message sent from the switch to the controller to inform about the removal of a flow entry from a flow table. This message carries all the statistics measured for the removed flow during its lifetime. Flags indicate the reason for removing the flow entry (*e.g.,* hard or idle timeout expiration).

- `OFPT_MULTIPART_REQUEST`: message sent from the controller to the switch to request measured statistics. To specifically request statistics from active flow entries of a given flow table, this message must by of type `OFPMP_FLOW`.

- `OFPT_MULTIPART_REPLY`: message sent from the switch to the controller containing requested statistics. This message will be of type `OFPMP_FLOW`, and consisting of statistics of active flow entries in a given table, when sent in response to an `OFPT_MULTIPART_REQUEST` of the same type.

Figure 6.2 shows the exchange of messages listed above between the OpenFlow controller and the switch, and the processes that trigger the messages exchange. Note that, for brevity's sake, this figure considers that the OpenFlow switch implements a single flow table (*i.e.,* table 0). However, according to the OpenFlow specification [91], the actions set of a flow entry might contain the command to search for another match in another flow table. The `OFPT_FLOW_MOD` message received by the switch in reply to an `OFPT_PACKET_IN` previously sent to the controller (blue lines in the figure) contains the specifications of the flow entry to be added to the flow table.

The process of checking for expired flow table entries, showed in Figure 6.2, is executed in a periodic fashion. The periodicity is, however, implementation dependent, *i.e.,* loop-time might differ depending on the OpenFlow implementation. In this cycle, the OpenFlow switch checks from time to time whether flow entries in any flow table have expired due to hard or idle timeouts. If any, the expired flow entries are removed and, provided that these had the flag `OFPFF_SEND_FLOW_REM` set, for each removed entry an `OFPT_FLOW_REMOVED` message is sent to the controller (red line in the figure) containing statistics of the removed entry.

The last interaction showed in Figure 6.2 is initiated by the OpenFlow controller when it sends an `OFPT_MULTIPART_REQUEST` message of type `OFPMP_FLOW`

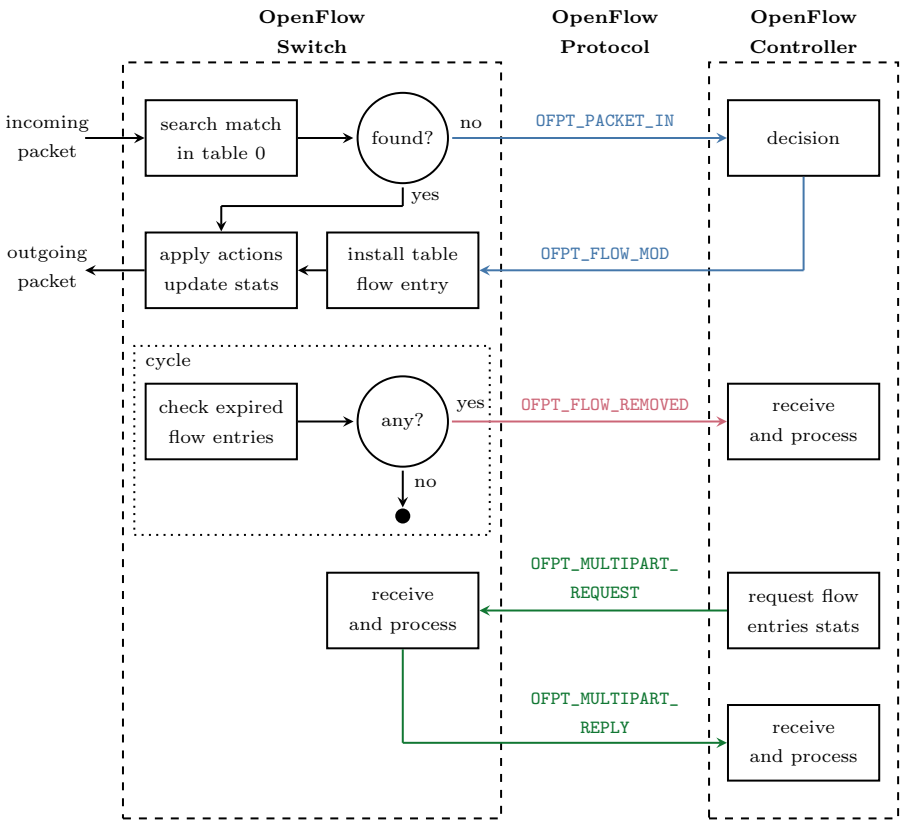


Figure 6.2: Messages exchange between OpenFlow controller and switch.

to the switch (green lines in the figure). This message should specify which flow table the request targets. Upon receiving this request, the switch replies to the controller with an `OFPT_MULTIPART_REPLY` message of the same type, containing current statistics of all active flow entries in the required table. The need for sending such request is determined by the application running on top of the controller and it is not defined by the OpenFlow specification (*i.e.*, it is not a periodic message exchange).

6.3 OpenFlow-based Approach

In this section we detail the proposed approach for using OpenFlow traffic measurements for link dimensioning purposes. As previously mentioned, OpenFlow provides some basic per-flow metrics, namely duration and number of packets and bytes. These are exactly the information we use from NetFlow/IPFIX style flows for the approaches proposed in Chapters 4 and 5. Therefore, in this chapter we do not propose a whole new link dimensioning approach, but rather we propose to combine OpenFlow traffic measurements and one of the previously proposed flow-based link dimensioning approaches. Due to its simplicity, we decided to combine the flow-based approach from Chapter 4 with OpenFlow measurements. Notice that *flow table entries* are the OpenFlow equivalent for the *flow records* of Chapter 4's flow-based approach.

> ***Recap:*** *the flow-based approach described in Section 4.2 estimates link required capacity from time series built solely based on flows. The process of building the flow-level time series assumes that the metered per-flow bytes are uniformly distributed within the flow record's duration. Therefore, the only metrics needed from OpenFlow traffic measurements are the duration of flow entries and the number of transferred bytes during their respective lifetime in the flow table.*

As shown in Figure 6.3, the proposed OpenFlow-based approach is a mix of passive and active operations using the messages described in the previous section. The *passive portion* of our approach is given by the information we are able to collect from `OFPT_FLOW_REMOVED` messages (red lines in the figure). That is, every time the controller receives an `OFPT_FLOW_REMOVED` message from the switch, the statistics about the flow are processed according to the approach described in Section 4.2.2. This is called passive portion because the approach does not initiate any process for receiving `OFPT_FLOW_REMOVED` messages. Instead, these messages are received due to OpenFlow operation.

The *active portion* of the proposed approach, blue lines in Figure 6.3, consists of a periodic request for statistics sent from the OpenFlow controller to the switch. This is called active portion because the process is initiated by our approach by sending the request to the switch using the `OFPT_MULTIPART_REQUEST` message with type `OFPMP_FLOW`. Upon receiving a reply from the switch, the statistics of all flow entries received within the `OFPT_MULTIPART_REPLY` are processed again following the approach described in Section 4.2.2. At this point, the link dimensioning approach might request to the switch to reset the counters of all active flow table entries. This would avoid the need of keeping track of active entries lasting since the previous cycle so that proper differences are calculated

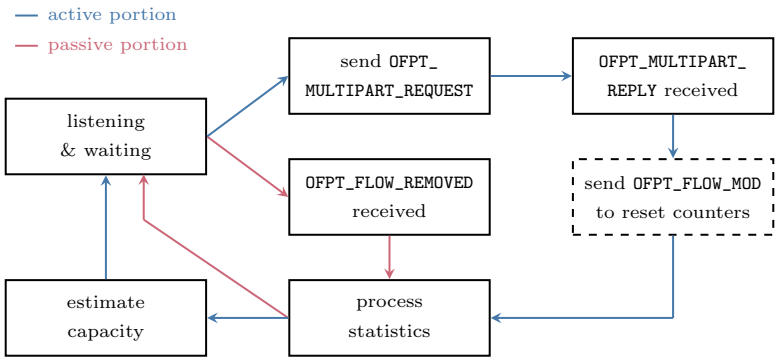


Figure 6.3: Proposed approach running on top of the OpenFlow controller.

(*i.e.,* statistics from the previous request are not accounted). Such approach, however, might significantly increase the amount of control messages between the OpenFlow controller and the switch, since for each active flow entry in the table, an `OFPT_FLOW_MOD` message with flag `OFPFF_RESET_COUNTS` must be sent. Nonetheless, we do not go further on this performance issue, and assume this to be an implementation decision. To complete the periodic cycle the link dimensioning approach computes the required capacity based on all the information passively or actively gathered from the switch. Finally the approach returns to the initial state for a new cycle.

It is worth of mentioning again that the proposed OpenFlow-based approach for link dimensioning assumes flows to be defined as in the case of NetFlow/IPFIX. More precisely, we define a flow with the 5-tuple as in NetFlow v5. That is, in our OpenFlow-based approach the *match fields* of flow table entries are defined by the following five fields: source and destination IP addresses, source and destination ports and transport protocol. In the next section we describe the experiments setup and present results of our preliminary assessment on the quality of OpenFlow traffic measurements.

Notice that the approach proposed in this section is theoretically possible due to operations and features specified in OpenFlow [91]. In practice, however, we observed that the quality of measurement data obtained from current implementations of OpenFlow is the limiting factor in the deployment of this approach. This has prevented us to validate the proposed OpenFlow-based approach. Nonetheless, in the next section, we assess the quality of OpenFlow traffic measurements, pointing out pitfalls and potential causes for the lack of accuracy of measured data.

6.4 OpenFlow Traffic Measurements

For the experiments described in this section we setup a virtualized and a physical OpenFlow environment that we used to assess the quality of traffic measurements provided by OpenFlow. In this section we first describe the experiments setup and then present results of our analysis of the quality of the traffic measurements obtained from OpenFlow protocol. Finally, we present a discussion on our findings, indicating potential reasons for the identified problems.

6.4.1 Environment Setup

In this section we detail the environment setup built to experiment on OpenFlow. We start by describing the application we implemented on top of Ryu OpenFlow controller. Then we detail both the physical and the virtual and setups used to study OpenFlow. We have used these two setups so that we were able to assess quality of OpenFlow measurements obtained from a real OpenFlow switch (physical setup) running a vendor specific implementation, and also from a virtual one using Open vSwitch (OVS)[1], which serves as basis for many vendor implementations of OpenFlow.

OpenFlow controller

Our implementation runs on top of Ryu[2] controller. Ryu is an open source implementation in Python of an OpenFlow controller and it is compatible to OpenFlow 1.3. This enabled us to already build our implementation and experiments using OpenFlow 1.3, an advantage compared to most of the available controller implementations that are still based on OpenFlow 1.0. Ryu implements the basic synchronization between controller and switch. Our implementation written on top of Ryu executes the tasks shown in Figure 6.3 (except the *estimate capacity* box). It processes the OpenFlow messages that are of our interest, *i.e.,* the messages described in Section 6.2.3. Our controller implementation is used for both physical and virtual setups, which are described next, and the code is publicly available at `https://github.com/ricardoschmidt/openflow`.

Physical setup

The physical setup consists of three servers and one switch. As shown in Figure 6.4 our OpenFlow controller runs on one of the servers. The server A is used as the traffic source. The server B is used to receive the traffic sent by A and forwarded by the OpenFlow switch. For each flow entry installed in the

[1] `http://openvswitch.org/`
[2] `http://osrg.github.io/ryu/`

switch, our OpenFlow controller uses the same action set, which is to forward flow packets from A to B. In this setup the communication only flows from A to B and the latter simply acts as a sink to receive forwarded traffic. At A we replay packet traces, previously captured, using the tool `tcpreplay`. Replaying previously captured traces allows us to repeat the experiments multiple times with the same input traffic. The OpenFlow switch in our physical setup is a Pica8 Pronto 3295[3], running PicOS 2.3, which is based on OVS 2.0.90. PicOS supports OpenFlow versions from 1.0 to 1.3.

An important remark is that `tcpreplay` gives us a summary of the replayed traffic. This allows us to check whether the numbers of packets and bytes received at B are the same as the actually numbers replayed at A. To check the received traffic at B we use `tcpdump`. If the numbers of packets and bytes seen at A and B match, it means that all packets were correctly forwarded by the OpenFlow switch and, hence, we expect to see those same numbers correctly reported in the OpenFlow metered statistics.

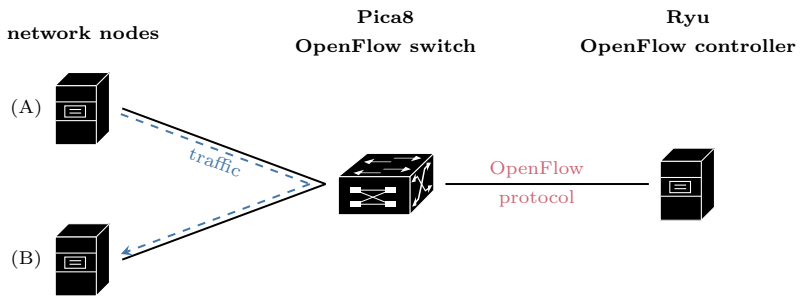


Figure 6.4: Physical environment setup.

Virtual setup

As opposed to the physical setup, in the virtual one we do not have an actual OpenFlow switch. The virtual setup is quite similar to the one presented in Figure 6.4. It consists of three virtual machines (running on the same physical machine): (i) one configured as an OpenFlow switch running OVS 2.1.2; (ii) one running our OpenFlow controller implementation; and (iii) one configured to act as the traffic sender, also using `tcpreplay` to replay packet traces. Running the original implementation of OVS in the virtual environment (*i.e.,* without extensions or modifications) allows us to assess the quality of OpenFlow

[3] `http://www.pica8.com/`

measurements without being biased by implementations of a specific vendor. Implementation decisions by different vendors, such as Pica8, might affect the OpenFlow performance, behavior and measurements.

6.4.2 Assessing the Quality of Measurement Data

In this section we present the results of our assessment on the quality of traffic measurements obtained from OpenFlow in both physical and virtual setups described above. For this experiments, we selected three flows[4] from a randomly chosen trace, from location F, from the dataset described in Chapter 2. The choice for these three specific flows was given by the fact that when replaying the whole trace in our virtual setup, these three flows were reported with significant amount of unaccounted packets and bytes. The three selected flows have distinct characteristics and are summarized in Table 6.1. Note that these flows were replayed individually and not as an aggregate.

Table 6.1: Flows used as input in our experiments.

flow	duration (s)	avg. rate	# of packets	# of bytes
1	104.64	277.12 kb/s	5056	3695617
2	300.16	73.51 kb/s	68464	4122630
3	18.79	13.89 kb/s	756	47118

The main goal of this experiment was to check whether different packet arrival rates affect the accuracy of OpenFlow counters. Clearly, we expect that higher packet rates might cause serious scalability problems at the OpenFlow controller due to the higher load of control traffic between switch and controller. Ultimately, this load might affect the quality of traffic measurements. The bottom part of Figure 6.5 shows the rate of packets sent every second for each of the three flows of Table 6.1, and the upper part of the same figure shows the flow entries/records exported by OVS (OpenFlow virtual setup) and NetFlow (for matters of comparison). Once again, to force the creation and exporting of multiple flow entries/records in OpenFlow/NetFlow, we set hard/active timeout to 15s and idle/inactive timeout to 3s. In the upper part of Figure 6.5, each circle indicates a `OFPT_FLOW_REMOVED` message sent by the OpenFlow switch to the controller reporting an expired flow entry or, in the case of NetFlow, an exported flow record also due to timeout expiration. The size of the circle is determined

[4] 5-tuple flow key defined by: source and destination IP addresses, source and destination ports and transport protocol.

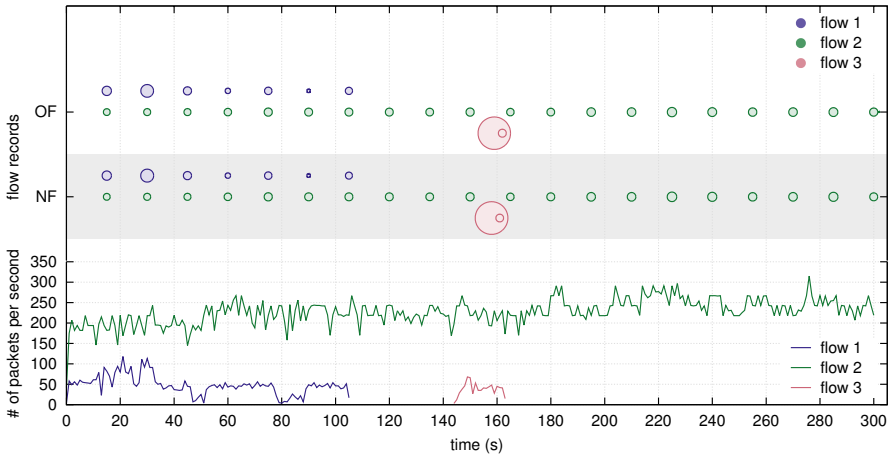


Figure 6.5: Bottom plot: number of packets transferred per second for each flow from Table 6.1. Upper plot: flow records for each flow using OpenFlow (OF) and NetFlow (NF). The size of circles indicates the relative amount of packets per flow record in relation to the total amount of packets in the flow. For OpenFlow, these numbers are an average of 20 runs per flow in the virtual setup.

by the relative number of packets accounted in that flow entry/record compared to the whole flow.

One can hardly see any difference between exported records in OpenFlow and NetFlow in Figure 6.5. This proves that OpenFlow can provide NetFlow-like traffic measurements. However, although very similar, there are some subtle inaccuracies in the measurements from OpenFlow. These can be seen in the numbers reported in Table 6.2. Although these inaccuracies seem to be minimal, we should keep in mind that only a single flow was sent through the OpenFlow switch at a time. Hence, inaccuracies might become meaningful when bigger traffic aggregates go through the switch. Notice that Table 6.2 shows the average and standard deviation of 20 runs for each flow from Table 6.1 in our virtual OpenFlow setup. Table 6.3 shows the average reported packets and bytes by Pica8 switch, in our physical OpenFlow setup. An important remark from the results in this table is that, besides byte counters being way off the actual sent traffic, one can clearly see that packets counter reported values 100 times less than the bytes counter.

As shown in Table 6.2, OVS reported in average a smaller amount of packets and bytes than actually sent to the switch. The results in this table, however, do not show that the reported statistics were quite random during the 20 runs for

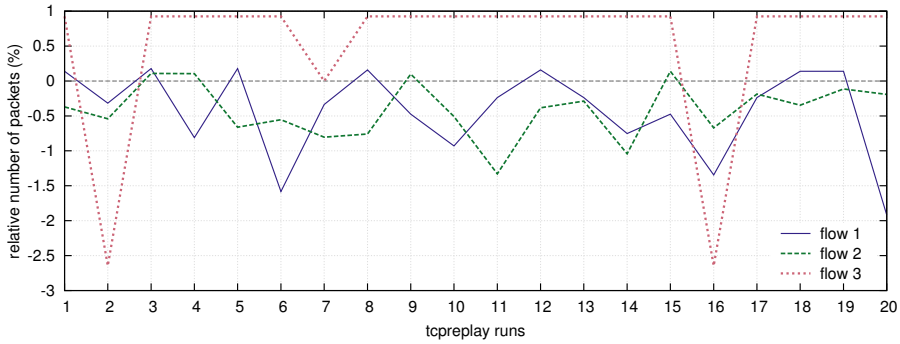
Table 6.2: Average and standard deviation (σ) of number of packets and bytes reported for each flow on 20 runs using the virtual setup.

flow	# of packets		# of bytes	
	avg	σ	*avg*	σ
1	5029.90	43.47	3674587.86	34897.98
2	68124.90	302.65	4102194.76	18221.35
3	754.00	13.04	46990.67	809.10

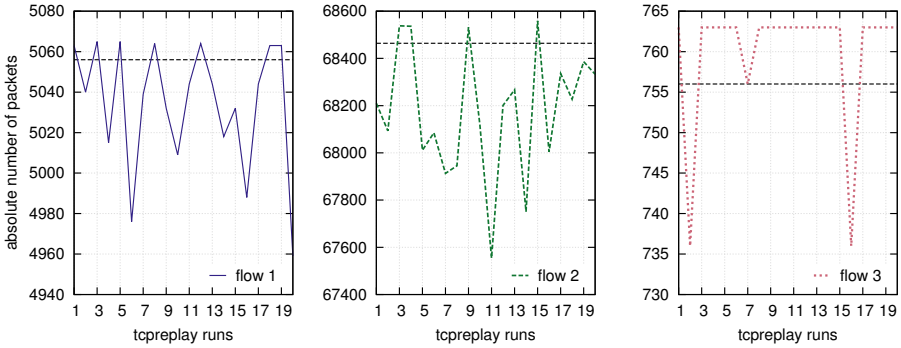
Table 6.3: Average and standard deviation (σ) of number of packets and bytes reported for each flow on 10 runs using the physical setup.

flow	# of packets		# of bytes	
	avg	σ	*avg*	σ
1	42626.00	14287.64	4283439.70	1437125.24
2	96399.90	30742.13	9689768.20	3096310.14
3	523.20	104.65	53458.90	10992.58

each flow. That is, although in most of the runs the reported number of packets and bytes was smaller than actual values in the flows, in few runs OVS surprisingly reported higher numbers of measured packets and bytes. Figure 6.6 shows how the number of reported packets varies from one execution of `tcpreplay` to another in our virtual setup. Although not shown in Figure 6.6, the number of bytes varies accordingly to the number of packets. Most of the time for *flow 1* and *flow 2* the reported number of packets by OVS is smaller than the actual one, with a maximum of about 2% less packets for *flow 1* (*i.e.,* 97 packets) and 1.5% less packets for *flow 2* (*i.e.,* 910 packets). However, in many runs, for all three flows, OpenFlow reported a higher amount of packets than actually sent. In these cases, for *flow 1* and *flow 2* the reported number of packets was never higher than 0.2%; that is 9 packets for *flow 1* and 95 packets for *flow 2*. For *flow 3*, however, most of the times OpenFlow reported around 1% more packets than actually sent (*i.e.,* 7 packets). Notice that results for *flow 3* might look worse than for *flow 1* and *flow 2*, but few missing packets have more impact in *flow 3* due to its smaller actual size as compared to the other two flows (see Table 6.1). For example, *flow 3* misses up to 2.6% of packets, which actually accounts for 20 packets only. An important observation is that numbers exported by OVS are quite random and in a second set of 20 runs of `tcpreplay`, we can expect number of exported packets to be completely different than what is shown in Figure 6.6. Another important observation from the results in Fig-



(a) Relative difference between actual number of packets in the flows (normalized to zero) and the reported number of packets by OVS.



(b) Absolute difference between actual number of packets in the flows and the reported number of metered packets by OpenFlow. The dashed lines show the actual number of packets for each flow.

Figure 6.6: Varying number of reported packets per flow across 20 runs of `tcpreplay` in the virtual setup.

ure 6.6 is that among all runs for all three flows (60 in total), only at run 7 the number of metered packets by OpenFlow was the same as the actual number of sent packets of one flow (*i.e., flow 3*).

6.4.3 Discussion

The problems for accounting less packets or bytes than actually sent to the OVS or Pica8 Pronto switch can arise from multiple reasons. First, high rates of packet arrivals might overload the controller or even the link connecting the

controller to the switch. This way `OFPT_PACKET_IN` messages might be lost due to software or hardware limitations. Packet loss might also be result of buffer overflow in the switch. At higher packets arrival rate, the switch might run out of space to store information about packets that triggered `OFPT_PACKET_IN` messages to the controller. This way, once the controller returns a `OFPT_FLOW_MOD` with instructions of what to do with the buffered packet, the switch might not have information about that packet anymore. A second reason for accounting less packets than actually sent might be related to implementation issues. The OpenFlow controller should be able to handle many `OFPT_PACKET_IN` messages arriving almost simultaneously. If it is not the case, the controller might drop such messages due to overload. The OpenFlow switch should also be able to handle many almost simultaneously `OFPT_FLOW_MOD` messages sent by the controller, install the new flow entries and account for the buffered packets.

OVS exploits parallelism and this might create situations in which an arriving packet is not accounted or even accounted twice due to lack of proper synchronization between parallel tasks. This can also explain situations in which OpenFlow reports more packets than actually sent.

Clearly, the language in which the OpenFlow controller is implemented can also determine its performance limitations. We have used a Python implementation of Ryu controller. Controller performance can be improved if it is implemented in more efficient language such as Java or C.

Differences or inaccuracies on flow duration and delay on "exporting" flow records can also be a result of the way OpenFlow is implemented. Both OVS and PicOS implement cycles for checking expired flows and for updating counters. For example, in OVS the cycle for checking expired flows due to hard or idle timeouts is performed every 2 seconds. Therefore, expired flows might have an additional duration of up to 2 seconds. According to the OVS documentation, the cycle for updating counters is performed every 1 second. That is, the latest packet(s) that arrived and matched with a flow entry might not be accounted if the hard timeout of its respective flow entry expires (and the entry is removed) before the next cycle for updating statistics is triggered.

We have observed another problem when replaying a whole 15-minute trace from our dataset in the physical setup. On having several hundreds of flow entries to manage, the OpenFlow switch seems to get lost in the measurement process and reports many *zeroed flows*. That is, flow records reported with counters set to zero. This problem only appears in the physical setup and if the number of simultaneously active flow records is "high enough" (note that a "high number" in our experiments was only few hundreds of simultaneously active flow records). All zeroed flows we have observed are flows that actually consist of a single packet. It is not easy to determine what might be the cause of this problem, but this is likely to be a result from high packet arrival rate combined

with the low priority given to the traffic measurement operations. That is, packets are correctly forwarded, but they might not be accounted because of overload in the OpenFlow switch.

6.5 Concluding Remarks

The growing interest from academia and industry on SDN motivated us to investigate the feasibility of deploying our flow-based link dimensioning approach in an OpenFlow-enabled network. That's because, among many other operations, OpenFlow meters some basic traffic statistics that suffice for link dimensioning tasks. In this chapter we proposed an approach to collect traffic measurements from the OpenFlow switch solely using messages defined by the OpenFlow protocol. This approach is a mix of active and passive techniques, where the OpenFlow controller (i) awaits for messages containing measurement data to be sent by the switch from time to time or (ii) explicitly requests for measurement data every time the bandwidth estimation is to be calculated. The flexibility of flow definition in OpenFlow allows us to define flows in the same way these are defined in our flow-based link dimensioning approaches from Chapters 4 and 5. That is, in theory OpenFlow can provide NetFlow/IPFIX style traffic measurements. Therefore, in this chapter we did not present a whole new way to compute traffic statistics required by dimensioning formula of Equation (1.2), but we propose to investigate the application of OpenFlow measurement data to one of our proposed flow-based approaches.

In this chapter we assessed the quality of OpenFlow per-flow traffic measurements and, although possible in theory, we have identified many problems with the accuracy of such measurements. The measurement operation of recently released OVS 2.1.2 and PicOS 2.3 (Pica8 Pronto 3295 switch) proved to be quite inaccurate and erratic. In both cases, OpenFlow did not provide us with accurate traffic measurements even when traffic load was absurdly low, *e.g.,* a single 500 kb/s flow. However, we observed that traffic was mostly correctly forwarded by the OpenFlow switch, which means that only the measurement operation was not performing as expected. This leads us to believe that measurement tasks are set as low priority at OpenFlow switch (*i.e.,* job priority), resulting in poor quality measurements. Moreover, additional inaccuracies on flow statistics seem to result from implementation decisions that can vary from vendor to vendor. For example, for an unknown reason to us, PicOS does not measure number of packets per flow, but estimates it from bytes counters. In OVS, as another example, the cycles for checking expired flows and for updating counters can affect the accuracy of, respectively, flow duration and packets/bytes counters.

Most of these problems can apparently be alleviated by improving OpenFlow implementations towards better quality measurements. It seems to be the case that OpenFlow community is not yet focused on making traffic measurement operations efficient and accurate. However, the growing adoption of OpenFlow will demand for such improvements. By the way, it is a fact that even well established technologies for traffic measurement also have problems, as reported in [106] for the case of SNMP and in [57] for the case of NetFlow. Nonetheless, there are few recent works, such as [85], that started to investigate how to improve the quality of OpenFlow measurements.

As further research, we plan to investigate even further the causes of inaccuracies in traffic measurements and, if possible, propose ways to correct them. Another clear next step is to implement a proof-of-concept of the OpenFlow-based link dimensioning approach proposed in Section 6.3 using OpenFlow measurement data and the flow-based approaches from Chapters 4 and 5. Our initial expectation is, however, that the link dimensioning approaches will call for further enhancements to account for the poor quality of measurement data from current OpenFlow implementation.

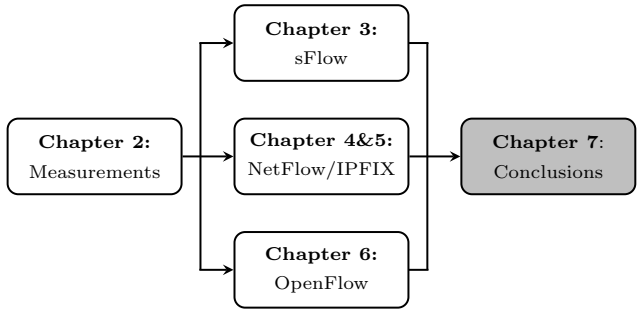
So long, and thanks for all the fish.

— Douglas Adams
In: The Hitchhiker's Guide to the Galaxy series.

CHAPTER 7

Conclusions

In this last chapter we set out the main findings of the research in this thesis, compare the proposed approaches for link dimensioning, and present directions for future research.



The organization of this chapter is as follows:

- *Section 7.1 summarizes the research presented in this thesis.*
- *Section 7.2 sets out the main conclusions of this thesis according to their respective research questions.*
- *Section 7.3 positions the proposed approaches for link dimensioning according to their respective easiness of use and accuracy.*
- *Section 7.4 summarizes the contributions of this thesis per chapter.*
- *Section 7.5 discusses directions for further research.*

7.1 Overview

The growing traffic demands combined with the trend towards virtualization of services and networks has widened the scope of link dimensioning approaches. Scenarios envisioned for the Future Internet demand more precise and intelligent link dimensioning to help (i) ensuring optimal use of bandwidth resources for multiple tenants and across multiple domains, and (ii) providing desired QoS

levels. On the one hand, the straightforward and widely used SNMP-based rules of thumb for link over-provisioning are easy-to-use, but not very precise, often resulting in a waste of link resources due to excessive over-estimation of required capacity. On the other hand, more advanced and accurate approaches for link dimensioning rely on continuous packet capturing and, hence, are difficult to use due to financial (*i.e.,* expensive devices) and operational (*i.e.,* not scalable to measured traffic) constraints. This motivated us to investigate and develop alternative approaches for link dimensioning with the properties of being – almost – as easy-to-use as SNMP-based rules of thumb and – almost – as accurate as packet-based approaches such as [109]. Therefore, we defined the goal of this thesis as

> *How can we accurately estimate required link capacity, for means of link dimensioning, with easy-to-use traffic measurement technologies?*

Aiming at ease of use and accuracy, in this thesis we proposed and analyzed four link dimensioning approaches. These approaches combine traffic measurement technologies that are largely found at network operators with the dimensioning formula proposed and validated in [109]. By using largely available technologies, we overcome the financial challenge of demanding expensive hardware/software for traffic measurements (as compared to continuous packet capturing). Initially, we performed an extensive study on properties of traffic from our measurements dataset. This study aimed at ensuring that current network traffic is still Gaussian distributed, which is one of the requirements of the adopted dimensioning formula from [109] originally proposed in 2006. Then we used our dataset to validate each of the proposed link dimensioning approaches in this thesis. In the next section we summarize the key findings of this thesis.

7.2 Main Conclusions

In this section we describe the main conclusions of this thesis according to their respective research questions defined in Chapter 1. In the first part of this thesis, the research question RQ-1, defined as

> *[...] is current Internet traffic still Gaussian?*

was addressed in Chapter 2. We have demonstrated that the emergence of new applications in the past years that reshaped users behavior, such as video streaming, online storage and social network services, has not significantly impacted on traffic Gaussianity fit. This allowed us to safely use the dimensioning formula from [109], which was built upon the assumption of traffic Gaussianity.

We also showed that, unlikely stated by previous works, it is sometimes safer to relate traffic Gaussianity to the measured traffic rate instead of the number of simultaneously active hosts. However, we demonstrated that, even among several thousands of simultaneously active hosts, very few hosts might have a negative effect on Gaussianity fit by sending a high volume of data with high rates using a single application/port.

In the second part of this thesis we proposed link dimensioning approaches that use traffic measurements technologies largely found at operator's infrastructures, namely sFlow, NetFlow/IPFIX and OpenFlow. Chapter 3 addressed the research question RQ-2, defined as

> *[...] how can we estimate traffic average rate and variance from sampled packets?*

We proposed and validated approaches to calculate traffic variance (ultimately used for estimating required capacity) from sampled data obtained from measurement technologies employing different packet sampling strategies. We validated the accuracy of our proposed approaches with data from three sampling strategies: Bernoulli, n-in-N and the specific strategy implemented by the sFlow tool. Besides being widely implemented by traffic measurement tools, Bernoulli and n-in-N are also defined in [119]. We demonstrated that for sampled data to be useful for link dimensioning, independently of the sampling strategy, a sampling rate should be chosen taking into consideration both the timescale of interest and the volume of traffic on the monitored link. From our measurements dataset we could not conclude to which extent traffic throughput plays a role on the sampling rate choice. Nonetheless, it is clear that higher sampling rates are mostly required at smaller timescales while lower sampling rates are mostly required at larger timescales, as shown in Figure 3.14. Another problem we addressed was the aggregation of sampled packets at the sFlow agent, which results on loss of individual packet timestamps. We showed that in the case of sFlow, the buffer size at the sFlow agent and the aggregation of sampled packets on the exporting process do not invalidate sampled data for link dimensioning given that a correct sampling rate is set. For example, for traces in our dataset with average rates 1 Gb/s, traffic sampled 1:100 was enough to estimate required capacity at timescales as low as 100ms using any of the three tested sampling algorithms.

The research question RQ-3 defined as

> *[...] how can we estimate traffic average rate and variance without information on individual packets?*

was addressed in Chapters 4 and 5. A purely flow-based approach for link dimensioning was proposed in Chapter 4. We demonstrated that this approach

can account for the lack of packet-level information and estimate required link capacity solely from flows at timescales around 1s. In Chapter 5, we proposed a hybrid approach in which the flow-based approach is combined with analytical models accounting for the packet-level dynamics within flows. Extensive validation experiments proved that it is possible to overcome the data aggregation problem in flows and, ultimately, obtain accurate estimations of required capacity at timescales as low as 1ms.

Lastly, in Chapter 6 we addressed the research question RQ-4 defined as

> *[...] can we use OpenFlow per-flow traffic measurements for link dimensioning purposes?*

We proposed an approach that uses the OpenFlow protocol to retrieve traffic measurements from the switch in a passive and active fashion. In theory such approach works. In practice, however, the lack of accuracy of OpenFlow traffic measurements impair the use of such measurements for link dimensioning purposes. From results collected in experiments with a virtual and with a physical OpenFlow setup, we assume that the lack of accuracy of per-flow measurements in OpenFlow might result from, among others: (i) implementation-specific decisions such as periodical cycles to check expiration of flow entries and update per-flow statistics; and (ii) scalability problems, such as capability of controller and switch to handle high packet arrival rates and high number of simultaneously active flows. Therefore, to achieve good quality traffic measurements in OpenFlow, we first need implementations of the protocol that give the right priority on implementing and executing measurement-related tasks.

7.3 Positioning of the Proposed Approaches

In Chapter 1, Figure 1.4 positioned the goal of this thesis in relation to the easy-to-use SNMP-based rules of thumb for link over-provisioning and the accurate packet-based approach for link dimensioning from [109]. Figure 1.4 indicated that our goal was to find a tradeoff between ease of use and accuracy. In this section, with a purely qualitative analysis, we position each of our proposed approaches for link dimensioning in the same plot, now depicted in Figure 7.1. Note that this figure does not capture the dimension of timescale in which estimations of required capacity would be done. Next, we explain the reason behind the positioning of each approach.

With the sFlow-based approach proposed in Chapter 3 we obtained very accurate estimations of required capacity, sometimes as accurate as the packet-based approach. Operational-wise, the sFlow-based approach, among all proposed approaches in this thesis, is likely the one that can be the easiest to use,

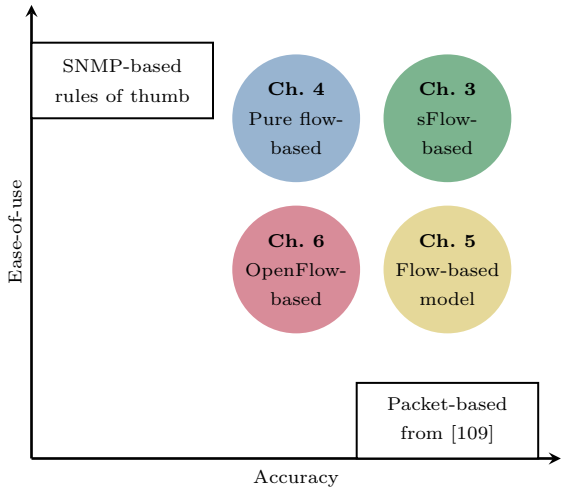


Figure 7.1: Position of the proposed link dimensioning approaches in this thesis.

but also the most difficult. What mainly defines the ease of use and accuracy in the sFlow-based approach is the chosen sampling rate. The lower the sampling rate the more scalable is the approach (easy-to-use), but it also becomes less accurate due to the small number of sampled packets. At the cost of lower scalability, higher sampling rates provide more measured data and, consequently, a better overview of traffic, which culminates in more accurate estimations of required capacity. There is, therefore, a trade-off between ease-of-use and accuracy mostly determined by the chosen sampling rate, and this justifies the position of the sFlow-based approach in Figure 7.1. Moreover, the fact that sFlow is largely available in today's network devices adds up to the ease of using the proposed sFlow-based approach for link dimensioning.

In Chapter 4 we proposed a flow-based link dimensioning approach that uses solely flows (NetFlow v5 style) to estimate required link capacity. Besides being widely available in network devices, flow measurements provide an aggregated view of the observed traffic and, hence, are more operationally scalable than plain packet capturing. Even if the network device does not have an embedded flow exporter, it is possible to use one of the many available pieces of software that provide NetFlow-like traffic measurements. The scalability of flows comes, however, at the cost of information loss due to the data aggregation. With (naive) general assumptions about traffic within flows, the approach proposed in Chapter 4 is able to accurately estimate required capacity at higher timescales only. This problem can be alleviated by manipulating the flows timeouts. By

reducing the timeouts, we can better capture short-term traffic fluctuations and, hence, estimate required capacity at smaller timescales. However, short timeouts are not typically deployed by operators. That is, on the one hand, setting shorter timeouts increases accuracy, but reduces the approach's ease of use. Clearly, on the other hand, using higher timeouts commonly set by operators (typically the default settings of the measurement technology) might reduce the approach's accuracy, but increases its ease of use. This trade-off added to the applicability limitation to higher timescales justifies Chapter 4's position in Figure 7.1.

The link dimensioning approach proposed in Chapter 5 aimed at overcoming the problem of the approach from Chapter 4. That is, how to accurately estimate required capacity at small timescales by using flow-level measurements. We combined a flow-based approach with an analytical model that compensates for the lower granularity of flow data, even when setting longer timeouts. Undoubtably, this model helped to improve the accuracy of the estimations at smaller timescales. However, the model requires fitting of parameters that can only be done using packet-level captures. Although only sporadically needed, these captures impose the need of an extra measurement effort as compared to the approach from Chapter 4. Therefore, the hybrid flow-based approach from Chapter 5 can be very accurate, sometimes as much as the packet-based approach, but scalability is impaired by the requirement of packet-level measurements, and this explains this approach's position in Figure 7.1.

Finally, in Chapter 6 we proposed an OpenFlow-based approach for link dimensioning. Many of the characteristics of the OpenFlow-based approach are very similar to the flow-based approach from Chapter 4, since the former combines the latter with OpenFlow per-flow traffic measurements. Note that the positioning of the OpenFlow-based approach here considers that in theory OpenFlow would be able to deliver per-flow measurements in a NetFlow/IPFIX style without the accuracy problems we presented in Chapter 6. If OpenFlow is able to deliver traffic measurements correctly, the accuracy of the estimations of required capacity will be very close to those obtained with the flow-based approach from Chapter 4. Although the adoption of OpenFlow is increasing, it is currently not as widely deployed as NetFlow and sFlow, or other variants of these. This low availability makes it more difficult to use than, *e.g.,* the pure flow-based approach or the sFlow-based approach.

All the proposed approaches have pros and cons. Drawbacks can mostly be alleviated by setting measurement parameters, such as sampling rate in sFlow and timeouts in NetFlow. However, one will always need to find a compromise between the desired accuracy and the ease of using of the adopted link dimensioning approach.

7.4 Summary of Contributions

In this section we list the main contributions of this thesis per chapter.

Chapter 2 – Datasets and Traffic Characteristics

- Demonstrates that it is safer to link Gaussianity fit of traffic to aggregated rate than to the number of active users generating traffic.
- Proves that the evolution of traffic led by the advent of new online applications and services, has not influenced Gaussianity fit of traffic.
- Reveals that the behavior of very few hosts and applications can negatively impact Gaussianity fit of traffic.
- Provides a public dataset consisting of anonymized packet traces of measurements performed by us (locations A, B and C of Table 2.1). The data comprises more than 48 hours of packet capturing carried out in 2011 and 2012. The traces are available at `http://www.simpleweb.org/wiki/Traces`.

Chapter 3 – sFlow-based Link Dimensioning

- Demonstrates that it is possible to use sampled packets for link dimensioning, giving accurate estimations of required capacity even at millisecond timescales.
- Reveals that, for link dimensioning purposes, the chosen sampling rate must take into consideration the traffic rate and the timescale of interest, and not only the link capacity, as suggested by the sFlow community.
- Evaluates the impacts of sFlow operations on the quality of measurements and their subsequent use for link dimensioning.
- Provides three packet sampling implementations that emulate Bernoulli, n-in-N and sFlow sampling algorithms. These implementations are public, under GPL v2 license[1], and can be downloaded at `https://github.com/ricardoschmidt/sampling`.

[1] `http://www.gnu.org/licenses/gpl-2.0.html`

Chapter 4 – Pure Flow-based Link Dimensioning

- Demonstrates the feasibility of estimating required link capacity at timescales around 1s by exclusively using flow-level traffic measurements (NetFlow v5 style flows).
- Provides an implementation of the proposed approach for creating flow-level time series, which can be later used for estimating traffic variance or other statistics. This implementations is public, under GPL v2 license, and can be downloaded at `https://github.com/ricardoschmidt/flow-ts`.

Chapter 5 – Hybrid Flow-based Link Dimensioning

- Extends the flow-level traffic models from [112] to better estimate traffic variance from flow-level measurements (NetFlow v5 style).
- Introduces packet-level models that account for packet behavior within flows.
- Demonstrates that by combining flow-level and packet-level models, accurate estimations of required capacity can be obtained even at millisecond timescales.

Chapter 6 – OpenFlow-based Link Dimensioning

- Proposes an approach to passively and actively retrieve traffic measurements from the OpenFlow switch using the OpenFlow protocol, for purposes of link dimensioning.
- Demonstrates that the quality of per-flow traffic measurements obtained from current implementations of OpenFlow lack accuracy required by link dimensioning.
- Provides an implementation of the OpenFlow controller used to retrieve measurement data from the OpenFlow switch, which can be later used for link dimensioning or other operations. This implementation is public, under GPL v2 license, and can be downloaded at `https://github.com/ricardoschmidt/openflow`.

7.5 Future Research

In this section we set out research directions to extend the work presented in this thesis.

- Further research can be conducted aiming at improving the accuracy of the proposed approaches for link dimensioning. In Chapter 3, for example, one could explore the use of additional statistics exported by the sFlow tool to support even better estimations of traffic average rate and variance from sampled data. In Chapters 4 and 5, it would be important to investigate to which extent sampled NetFlow would reduce the accuracy of the proposed link dimensioning and, if significant, how we could overcome this problem.

- In this thesis we investigated and developed approaches for link dimensioning. An obvious next step would be to implement and deploy these proposed approaches in actual network management applications, in experimental and/or production networks. Example of applications are:

  - *Energy-aware traffic engineering.* There are many works proposing energy-aware solutions, and they could certainly profit from link dimensioning. For example, in [50], we exploited the opportunistic use of MPLS backup paths to distribute traffic across the network and save energy by putting extra links to sleep. In [20] the authors proposed an energy-aware approach to periodically redistribute IP traffic within a core network, by setting idle line cards into sleep mode. These approaches usually (re)allocate link resources and redirect traffic based on average link utilization and, therefore, might end up impairing QoS. Link dimensioning can support the proper reallocation of resources for redirected traffic. Ultimately, gains on energy savings might be limited due to higher bandwidth requirements imposed by the link dimensioning as compared to average link utilization metrics. Future work in this area can assess the pros and cons of employing link dimensioning on approaches for energy-aware traffic engineering.
  - *Data center networks.* Data center networks are said to be transparent to tenants, who hire services based on computational and storage resources to be used and are charged on a time-of-use basis. However, data center networks are typically highly oversubscribed, which might create bottlenecks between the topology layers [11]. Ultimately, this might result in longer completion time of tenants' jobs, elevating costs of hired resources. Past works proposed to include bandwidth as part of tenants resource request and, consequently, as

part of resources allocation algorithms within the data centers. To the best of our knowledge, none of the past works have explored how link dimensioning can support resources management operations. So, as we also envisioned in [47], a topic for further research is to investigate how link dimensioning can assist allocation of link resources in data centers, aiming at optimal use of the data center network and fair pricing to tenants.

- OpenFlow is a very recent technology and there is still lots of room for improvement. In this thesis we provided a first investigation on the quality of per-flow traffic measurements in OpenFlow. We showed that although in theory possible by the OpenFlow standard [91], current implementations of OpenFlow do not provide measurement data of enough quality for link dimensioning purposes. These problems seem to be mainly caused by implementation decisions that potentially vary from vendor to vendor. It would be of great value to perform a more comprehensive study on the quality of OpenFlow traffic measurements, comparing implementations of different vendors. Such study would certainly pave the way for further improvements on OpenFlow, and network management could ultimately rely on traffic measurements obtained via this technology.

- The scenario of the future Internet, as detailed in Chapter 1, brings many multidisciplinary challenges. In this thesis we addressed the link dimensioning operations that can help on decisions for allocation of bandwidth resources. Other areas certainly deserve attention as well. For example, research should be conducted to explore security-related problems that might arise with virtualization of services and networks. Another direction that deserves attention and further investigation is on the ethical aspects of the Internet ecosystems, mainly concerning end users privacy and their relationship with the Internet big players.

Estimating Variance from Sampled Packets

In this appendix we show the derivation of our proposed formulas for estimating the traffic variance from sampled packets, as presented in Section 3.4.

A.1 Estimating Traffic Variance with Bernoulli Sampling

Let μ and $v(T)$ be the mean and the variance of the observed traffic per time interval before sampling. Under the assumptions given in Section 3.4.1, it holds

$$\mu = E[P]E[S] \,, \tag{A.1}$$

$$v(T) = E[P]Var[S] + E[S]^2 Var[P] \,. \tag{A.2}$$

Let μ' and $v'(T)$ be the mean and the variance of the sampled traffic per time interval. The "trick" with the zero-size packets described in Section 3.4.1 gives

$$\begin{aligned} \mu' &= E[P]E[S'] && \text{(A.3)} \\ &= pE[P]E[S] = p\mu \,, && \text{(A.4)} \end{aligned}$$

$$\begin{aligned} v'(T) &= E[P]Var[S'] + E[S']^2 Var[P] && \text{(A.5)} \\ &= p^2 E[P]Var[S] + \left(\frac{1}{p} - 1\right) E[S^2] + p^2 E[S]^2 Var[P] && \text{(A.6)} \\ &= p^2 \left(v(T) + \left(\frac{1}{p} - 1\right) E[P]E[S^2]\right) \,. && \text{(A.7)} \end{aligned}$$

The last equation can be solved for $v(T)$. With $r = 1/p$ and $v_{est}(T) = r^2 v'(T)$, we obtain the desired estimation of $v(T)$ shown in Equation (3.7).

A.2 Estimating Traffic Variance with 1-in-N and sFlow Sampling

Again, let μ and $v(T)$ be the mean and the variance of the observed traffic. As explained in Section 3.4.2, we assume that the number P'_i of sampled packets in time interval i is i.i.d. as P' with $P' = P/N$, where P is the number of packets per time interval before sampling. Thus, the mean μ' and the variance $v'(T)$ of the sampled traffic per time interval are

$$\begin{aligned} \mu' &= E[P']E[S] && \text{(A.8)} \\ &= \frac{1}{N}E[P]E[S] = \frac{\mu}{N}\,, && \text{(A.9)} \end{aligned}$$

$$\begin{aligned} v'(T) &= E[P']Var[S] + E[S]^2 Var[P'] && \text{(A.10)} \\ &= \frac{1}{N}E[P]Var[S] + \frac{1}{N^2}E[S]^2 Var[P] && \text{(A.11)} \\ &= \frac{1}{N^2}\left(v(T) + (N-1)E[P]Var[S]\right)\,. && \text{(A.12)} \end{aligned}$$

By solving for $v(T)$ and using $r = N$ and $v_{est}(T) = r^2 v'(T)$, we obtain the desired estimation of $v(T)$ shown in Equation (3.9).

Variance from flows with constant duration

This appendix shows the derivation of the formula to estimate traffic variance from flows assuming constant flow durations (Equation (5.3)). From the basic variance formula Equation (5.1), the derivation of the variance formula for *constant* flows duration D is as follows:

$$v(T) = \lambda r^2 \left(2T \int_0^T x(1 - F_D(x))dx - \delta \int_0^T x^2 f_{D^r}(x)dx + \delta T^2 (1 - F_{D^r}(T)) \right)$$

Splitting the equation above in some parts, we have:

$$\begin{aligned} A &= \int_0^T x(1 - F_D(x)dx \\ B &= \int_0^T x^2 F_{D^r}(x)dx \\ C &= 1 - F_{D^r}(T) \\ F_D(x) &= \begin{cases} 0 & \text{, if } T < c \\ 1 & \text{, if } T \geq c \end{cases} \end{aligned}$$

Solving A, B and C individually, we obtain:

$$\begin{aligned} A &= \int_0^T x dx - \int_0^T x F_D(x)dx \\ &= \begin{cases} \frac{T^2}{2} & \text{, if } T < c \\ \frac{T^2}{2} - \left(\frac{T^2}{2} - \frac{c^2}{2} \right) & \text{, if } T \geq c \end{cases} \end{aligned}$$

$$\begin{aligned} B &= \frac{1}{\delta} \int_0^T x^2 (1 - F_D(x))dx \\ &= \begin{cases} \frac{1}{\delta} \frac{T^3}{3} & \text{, if } T < c \\ \frac{1}{\delta} \left(\frac{T^3}{3} - \left(\frac{T^2}{2} - \frac{c^3}{3} \right) \right) = \frac{1}{\delta} \frac{c^3}{3} & \text{, if } T \geq c \end{cases} \end{aligned}$$

$$\begin{aligned} C &= 1 - F_{D^r}(T) \\ &= 1 - \int_0^T F_{D^r}(x)dx \\ &= 1 - \frac{1}{\delta}\int_0^T (1 - F_D(x))dx \\ &= \begin{cases} 1 - \frac{T}{\delta} & \text{, if } T < c \\ 1 - \frac{c}{\delta} & \text{, if } T \geq c \end{cases} \end{aligned}$$

For $T < c$, $v(T)$ becomes:

$$\begin{aligned} v(T) &= \lambda r^2 (T^3 - \frac{T^3}{3} + \delta T^2 - T^3) \\ &= \rho r \left(T^2 - \frac{T^3}{3\delta} \right) \end{aligned}$$

And for $T \geq c$, given that $c = \delta$, $v(T)$ becomes:

$$\begin{aligned} v(T) &= \lambda r^2 \left(2T\frac{c^2}{2} - \frac{c^3}{3} + \delta T^2 - T^2 c \right) \\ &= \rho r \left(T\frac{c^2}{\delta} - \frac{c^3}{3} + \delta T^2 - T^2 c \right) \\ &= \rho r \left(T\delta - \frac{\delta^2}{3} \right) \end{aligned}$$

Finally, the variance $v(T)$ becomes (Equation (5.3)):

$$v(T) = \begin{cases} \rho r \left(T^2 - \frac{T^3}{3\delta} \right) & \text{, if } T < \delta \\ \rho r \left(T\delta - \frac{\delta^2}{3} \right) & \text{, if } T \geq \delta \end{cases}$$

Flow models for different packet arrival processes

In this appendix we show the derivation of our proposed formulas for the packet correction factor to support flow-based link dimensioning, as presented in Section 5.

C.1 Flow model with poisson packet arrival

In this section we prove Equation (5.6) under the assumption that packet arrivals inside a flow are Poisson with rate μ and have arbitrary sizes i.i.d. like S.

Let T be the aggregation timescale. Obviously, if we observe the active traffic flows during an interval of length T, we will mostly only see fragments of those flows because most flows start or end outside that interval. Let L be the total length of flow fragments observed in that interval. Since packet arrivals are Poisson, the p.m.f. of the number of packets K_l arriving during that interval for $L = l$ is (neglecting packet transmission times)

$$\mathbb{P}[K_l = k] = \frac{(\mu l)^k}{k!} e^{-\mu l}.$$

Let S_i be the size of the ith packet, i.i.d. like S. The total traffic A_l arriving in that arrival is then

$$A_l = \sum_{i=1}^{K_l} S_i$$

with Laplace-Stieltjes transform (LST)

$$\begin{aligned} L_{A_l}(q) &= \mathbb{E}[e^{-q\sum_{i=1}^{K_l} S_i}] \\ &= \sum_{k=0}^{\infty} \frac{(\mu t)^k}{k!} e^{-\mu l} \left(\mathbb{E}[e^{-qS}]\right)^k \\ &= e^{-\mu l(1-\mathbb{E}[e^{-qS}])}. \end{aligned}$$

Let $g(l)$ be the p.d.f. of L, with LST $G(\cdot)$. The LST of the total amount of traffic $A(T)$ arriving in that interval is then

$$\begin{aligned} L_{A(T)}(q) &= \int_{l=0}^{\infty} g(l) L_{A_l}(q) dl \\ &= \int_{l=0}^{\infty} g(l) e^{-\mu l(1-\mathbb{E}[e^{-qS}])} dl \\ &= G(-\mu(1-\mathbb{E}[e^{-qS}])). \end{aligned}$$

Hence, the first and second moment of $A(T)$ are given by

$$\mathbb{E}[A(T)] = -\frac{d}{dq} G(-\mu(1-\mathbb{E}[e^{-qS}]))\Big|_{q=0} \tag{C.1}$$

$$\mathbb{E}[A(T)^2] = \frac{d^2}{dq} G(-\mu(1-\mathbb{E}[e^{-qS}]))\Big|_{q=0} \tag{C.2}$$

By applying basic differentiation rules to Equation (C.1) and using the identities

$$-\frac{d}{dx} G(x)\Big|_{x=0} = \mathbb{E}[L] \quad \text{and} \quad -\frac{d}{dx} \mathbb{E}[e^{-qS}]\Big|_{x=0} = \mathbb{E}[S]$$

we obtain

$$\mathbb{E}[A(T)] = \mu \mathbb{E}[S]\mathbb{E}[L].$$

Similarly, Equation (C.2) gives

$$\mathbb{E}[A(T)^2] = \mu \mathbb{E}[S^2]\mathbb{E}[L] + \mu^2 \mathbb{E}[S]^2 \mathbb{E}[L^2].$$

Hence, the variance of $A(T)$ is

$$\mathrm{Var}[A(T)] = \mu \mathbb{E}[S^2]\mathbb{E}[L] + \mu^2 \mathbb{E}[S]^2 \mathbb{E}[L^2] - \mu^2 \mathbb{E}[S]^2 \mathbb{E}[L]^2.$$

Noting that the difference $\mu^2 \mathbb{E}[S]^2 \mathbb{E}[L^2] - \mu^2 \mathbb{E}[S]^2 \mathbb{E}[L]^2$ is simply the variance $v_{flow}(T)$ of the traffic in the constant-traffic-rate model in Equation (5.1), introduced in Section 5.2, it holds

$$\mathrm{Var}[A(T)] = v_{flow}(T) + \mu \mathbb{E}[S^2]\mathbb{E}[L].$$

Using $\mu \mathbb{E}[S]\mathbb{E}[L] = \rho T$, where ρ is the mean of the total traffic throughput, we finally obtain Equation (5.6).

C.2 Flow model with bursty packet arrival

Let P the number of packets in a burst. Since $\mathbb{P}[P = i] = (1-p)^{i-1}p$, the mean and variance of P are $\mathbb{E}[P] = 1-p$ and $\mathrm{Var}[P] = (1-p)/p^2$, respectively. The byte size of the burst can be expressed as a sum of a random number of random variables: $S' = \sum_{i=1}^{P} S_i$. Under the assumption of S_i i.i.d. like S and the independence of S_i and P, it is known for such sums that

$$\begin{aligned} \mathbb{E}[S'] &= \mathbb{E}[P]\mathbb{E}[S], \\ \mathrm{Var}[S'] &= \mathbb{E}[P]\mathrm{Var}[S] + \mathbb{E}[S]^2\mathrm{Var}[P], \end{aligned}$$

which allows to calculate $\mathbb{E}[S'^2]$.

Bibliography

[1] EU FP7 Mobile Cloud Networking. `http://www.mobile-cloud-networking.eu/`. Online. Accessed Jan. 2014.

[2] EU FP7 UniverSelf (257513). `http://www.univerself-project.eu/`. Online. Accessed Jan. 2014.

[3] Iperf2. `http://iperf.sourceforge.net/`. Online. Accessed Feb. 2014.

[4] Multi Router Traffic Grapher. `http://oss.oetiker.ch/mrtg/`. Online. Accessed Dec. 2013.

[5] nfdump. `http://nfdump.sourceforge.net/`. Online. Accessed Feb. 2014.

[6] Round Robin Database Tool. `http://oss.oetiker.ch/rrdtool/`. Online. Accessed Aug. 2014.

[7] tcpdump. `http://www.tcpdump.org/`. Online. Accessed Feb. 2014.

[8] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., and Chou, W. 2014. A roadmap for traffic engineering in SDN-OpenFlow networks. *Elsevier Computer Networks 71*, 1–30.

[9] Anjum, B., Perros, H., Mountrouidou, X., and Kontovasilis, K. 2011. Bandwidth allocation under end-to-end percentile delay bounds. *International Journal of Network Management 21,* 5, 536–547.

[10] Barakat, C., Thiran, P., Iannaccone, G., Diot, C., and Owezarski, P. 2003. Modeling Internet backbone traffic at the flow level. *IEEE Transactions on Signal Processing 51,* 8, 1–12.

[11] Benson, T., Akella, A., and Maltz, D. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference*. IMC'10. 267–280.

[12] Bonald, T., Oliver, P., and Roberts, J. 2003. Dimensioning high speed IP access networks. In *Proceedings of the 8th International Teletraffic Congress*. ITC'03. 241–251.

[13] Brauckhoff, D., Tellenbach, B., Wagner, A., May, M., and Lakhina, A. 2006. Impact of Packet Sampling on Anomaly Detection Metrics. In *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference*. IMC'06. 159–164.

[14] Broido, A., Hyun, Y., Gao, R., and kc claffy. 2004. Their Share: Diversity and Disparity in IP Traffic. In *Proceedings of the 5th International Passive and Active Network Measurement Workshop*. PAM'04. 113–125.

[15] Brown, B. M. and Hettmansperger, T. P. 1996. Normal Scores, Normal Plots and Tests for Normality. *Journal of the American Statistical Association 91,* 436, 1668–1675.

[16] Cai, Z., Cox, A. L., and Ng, T. S. E. 2011. Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane. Tech. Rep. TR11-07, Rice University.

[17] CAIDA. The CAIDA UCSD Anonymized Internet Traces 2011 – 2011-05-19, 2011-07-21, 2011-12-22. `http://www.caida.org/data/passive/passive_2011_dataset.xml`. Online. Accessed Dec. 2013.

[18] CAIDA. The CAIDA UCSD Anonymized Internet Traces 2012 – 2012-01-19, 2012-02-16. `http://www.caida.org/data/passive/passive_2012_dataset.xml`. Online. Accessed Dec. 2013.

[19] CERT/NetSA. YAF – Yet Another Flowmeter. `http://tools.netsa.cert.org/yaf/`. Online. Accessed Feb. 2014.

[20] Charalambides, M., Tuncer, D., Mamatas, L., and Pavlou, G. 2013. Energy-Aware Adaptive Network Resource Management. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*. IM'13. 369–377.

[21] Choi, B.-Y., Park, J., and Zhang, Z.-L. 2003. Adaptive Random Sampling for Traffic Load Measurement. In *Proceedings of the 38th IEEE International Conference on Communications*. ICC'03. 1552–1556.

[22] Choi, B.-Y., Park, J., and Zhang, Z.-L. 2004. Adaptive Packet Sampling for Accurate and Scalable Flow Measurement. In *Proceedings of the 47th IEEE Global Telecommunications Conference*. GLOBECOM'04. 1448–1452.

[23] Chowdhury, S. R., Bari, M. F., Ahmed, R., and Boutaba, R. 2014. PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks. In *Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 1–9.

[24] Cisco Systems Inc. Cisco IOS Switching Services Configuration Guide, Release 12.2. `http://www.cisco.com/c/en/us/td/docs/ios/12_2/switch/configuration/guide/fswtch_c.html`. Online. Accessed Mar. 2014.

[25] Cisco Systems Inc. NetFlow Export Datagram Formats. `http://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.pdf`. Online. Accessed Mar. 2014.

[26] Cisco Systems Inc. 2005a. How To Calculate Bandwidth Utilization Using SNMP. http://www.cisco.com/image/gif/paws/8141/calculate_bandwidth_snmp.pdf. Online. Accessed Apr. 2014.

[27] Cisco Systems Inc. 2005b. Random Sampled NetFlow. http://www.cisco.com/c/en/us/td/docs/ios/12_0s/feature/guide/nfstatsa.pdf. Online. Accessed Mar. 2014.

[28] Cisco Systems Inc. 2007. NetFlow Services Solutions Guide. http://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/netflow/nfwhite.pdf. Online. Accessed May 2014.

[29] Cisco Systems Inc. 2008a. Cisco IOS Flexible NetFlow Configuration Guide – Release 12.4T. http://www.cisco.com/en/US/docs/ios/fnetflow/configuration/guide/12_4t/fnf_12_4t_book.html. Online. Accessed Dec. 2013.

[30] Cisco Systems Inc. 2008b. Cisco IOS Flexible NetFlow Technology. http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/flexible-netflow/product_data_sheet0900aecd804b590b.pdf. Online. Accessed May 2014.

[31] Cisco Systems Inc. 2013. Best Practices in Core Network Capacity Planning. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/quantum/white_paper_c11-728551.pdf. Online. Accessed Aug. 2014.

[32] Claise, B. 2004. Cisco Systems NetFlow Services Export Version 9. RFC 3954.

[33] Claise, B., Dhandapani, G., Aitken, P., and Yates, S. 2011. Export of Structured Data in IP Flow Information Export (IPFIX). RFC 6313.

[34] Claise, B. and Trammell, B. 2013. Information Model for IP Flow Information Export (IPFIX). RFC 7012.

[35] Claise, B., Trammell, B., and Aitken, P. 2013. Specifications of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011.

[36] Dainotti, A., Pescapé, A., and Claffy, K. C. 2012. Issues and Future Directions in Traffic Classification. *IEEE Network 26,* 1, 35–40.

[37] de O. Schmidt, R., Hendriks, L., Pras, A., and van der Pol, R. 2014a. OpenFlow-based Link Dimensioning. In *Proceedings of the Innovating the Network for Data-Intensive Science Workshop (INDIS), ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC'14.

[38] de O. Schmidt, R. and Pras, A. 2011. Estimating bandwidth requirements using flow-level measurements. In *Proceedings of the 5th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, PhD Workshop*. AIMS'11. 169–172.

[39] de O. Schmidt, R., Pras, A., and Gomes, R. 2011. On the Evaluation of Self-Addressing Strategies for Ad-Hoc Networks. In *Proceedings of the 17th International Workshop EUNICE*. EUNICE'11. 31–42.

[40] de O. Schmidt, R., Sadre, R., Melnikov, N., Schönwälder, J., and Pras, A. 2014b. Linking Network Usage Patterns to Traffic Gaussianity Fit. In *Proceedings of the 13th IFIP Networking Conference*. Networking'14. 1–9.

[41] de O. Schmidt, R., Sadre, R., and Pras, A. 2013a. Gaussian Traffic Revisited. In *Proceedings of the 12th IFIP Networking Conference*. Networking'13. 1–9.

[42] de O. Schmidt, R., Sadre, R., Sperotto, A., and Pras, A. 2013b. Lightweight Link Dimensioning using sFlow Sampling. In *Proceedings of the 9th International Conference on Network and Services Management*. CNSM'13. 152–155.

[43] de O. Schmidt, R., Sadre, R., Sperotto, A., and Pras, A. 2014c. Impact of Packet Sampling on Link Dimensioning. *Submitted to IEEE Transactions on Network and Service Management*, 1–13.

[44] de O. Schmidt, R., Sadre, R., Sperotto, A., van den Berg, H., and Pras, A. 2014d. A Hybrid Procedure for Efficient Link Dimensioning. *Elsevier Computer Networks 67*, 252–269.

[45] de O. Schmidt, R., Sperotto, A., Sadre, R., and Pras, A. 2012a. Towards Bandwidth Estimation using Flow-Level Measurements. In *Proceedings of the 6th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security*. AIMS'12. 127–138.

[46] de O. Schmidt, R., Sperotto, A., Sadre, R., and Pras, A. 2012b. Towards Bandwidth Estimation using Flow-level Measurements. In *Student Poster at TERENA Networking Conference*. TNC'12.

[47] Drago, I., de O. Schmidt, R., Hofstede, R., Sperotto, A., Karimzadeh, M., Haverkort, B. R., and Pras, A. 2013. Networking for the Cloud: Challenges and Trends. *Praxis der Informationsverarbeitung und Kommunikation, Special Issue on Data Center Networking 36,* 4, 207–214.

[48] Drago, I., Mellia, M., Munafò, M. M., Sperotto, A., Sadre, R., and Pras, A. 2012. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the ACM Internet Measurement Conference*. IMC'12. 481–494.

[49] Fraleigh, C., Tobagi, F., and Diot, C. 2003. Provisioning IP Backbone Networks to Support Latency Sensitive Traffic. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications*. INFOCOM'03. 375–385.

[50] Francois, F., Wang, N., Moessner, K., Georgoulas, S., and de O. Schmidt, R. 2014. Leveraging MPLS Backup Paths for Distributed Energy-Aware Traffic Engineering. *IEEE Transactions on Network and Service Management 11,* 2, 235–249.

[51] García-Dorado, J. L., Finamore, A., Mellia, M., Meo, M., and Munafò, M. M. 2012. Characterization of ISP Traffic: Trends, User Habits, and Access Technology Impact. *IEEE Transactions on Network and Service Management 9,* 2, 142–155.

[52] Gehlen, V., Finamore, A., Mellia, M., and Munafò, M. M. 2012. Uncovering the Big Players of the Web. In *Proceedings of the 4th International Workshop on Traffic Monitoring and Analysis*. TMA'12. 15–28.

[53] Google. Iperf3. `https://code.google.com/p/iperf/`. Online. Accessed Feb. 2014.

[54] Haag, P. 2006. NetFlow Tools NfSen and NFDUMP. In *Proceedings of the 18th Annual FIRST Conference*. FIRST'06.

[55] Hellemons, L., Hendriks, L., Hofstede, R., Sperotto, A., Sadre, R., and Pras, A. 2012. SSHCure: A Flow-Based SSH Intrusion Detection System. In *Proceedings of the 6th International Conference on Autonomous Infrastructure, Management and Security*. AIMS'12. 86–97.

[56] Hofstede, R., Bartos, V., Sperotto, A., and Pras, A. 2013a. Towards Real-Time Intrusion Detection for NetFlow and IPFIX. In *Proceedings of the 9th International Conference on Network and Service Management*. CNSM'13. 227–234.

[57] Hofstede, R., Drago, I., Sperotto, A., Sadre, R., and Pras, A. 2013b. Measurement Artifacts in NetFlow Data. In *Proceedings of the 14th International Conference on Passive and Active Measurement*. PAM'13. 1–10.

[58] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., and Pras, A. 2014. Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials*, 1–29.

[59] Hulboj, M. M. and Jurga, R. E. 2009. CERN Investigation of Network Behaviour and Anomaly Detection. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*. RAID'09. 353–354.

[60] IEEE. IEEE Standard for Local and metropolitan area networks – Link Aggregation. `http://tele.sj.ifsc.edu.br/~msobral/IER/802.1AX-2008.pdf`. Online. Accessed Jan. 2014.

[61] Inacio, C. M. and Trammel, B. 2010. YAF: Yet Another Flowmeter. In *Proceedings of the 24th Large Installation System Administration Conference*. LISA'10. 1–12.

[62] InMon Corp. 2003. sFlow Agent Software Description. `http://www.inmon.com/technology/InMon_Agentv5.pdf`. Online. Accessed Dec. 2013.

[63] Jarschel, M. and Pries, R. 2012. An OpenFlow-Based Energy-Efficient Data Center Approach. In *Proceedings of the ACM SIGCOMM conference*. SIGCOMM'12. 87–88.

[64] Jarschel, M., Zinner, T., Höhn, T., and Tran-Gia, P. 2013. On the Accuracy of Leveraging SDN for Passive Network Measurements. In *Proceedings of the Australasian Telecommunication Networks and Applications Conference (ATNAC)*. 41–46.

[65] Jasinska, E. 2006. sFlow: I can feel your traffic. In *Proceedings of the 23rd Chaos Communication Congress*. 23C3. 1–8.

[66] Jiang, H. and Drovolis, C. 2003. Source-Level IP Packet Bursts: Causes and Effects. In *Proceedings of the the 3rd ACM SIGCOMM Conference on Internet Measurement*. IMC'03. 301–306.

[67] Johnston, W. E., Dart, E., Ernst, M., and Tierney, B. 2013. Enabling high throughput in widely distributed data management and analysis systems: Lessons from the LHC. In *Proceedings of the TERENA Networking Conference*. TNC'13. 1–19.

[68] Jose, L., Yu, M., and Rexford, J. 2011. Online Measurement of Large Traffic Aggregates on Commodity Switches. In *Proceedings of the Workshop on Hot Topics in Management of Internet Cloud, and Enterprise Network and Services*. Hot-ICE'11. 1–6.

[69] Juniper Networks. 2011. Juniper Flow Monitoring. `http://www.juniper.net/us/en/local/pdf/app-notes/3500204-en.pdf`. Online. Accessed Jun. 2014.

[70] Kekely, L., Puš, V., and Kořenek, J. 2014. Software Defined Monitoring of Application Protocols. In *Proceedings of the 33rd Annual IEEE International Conference on Computer Communications*. INFOCOM'14. 1–9.

[71] Kilpi, J. and Norros, I. 2002. Testing the Gaussian approximation of aggregate traffic. In *Proceedings of the 2nd ACM SIGCOMM Internet Measurement Workshop*. IMW'02. 49–61.

[72] Klemm, A., , Lindemann, C., and Lohmann, M. 2003. Modeling IP Traffic Using the Batch Markovian Arrival Process. *Performance Evaluation 54*, 149–173.

[73] Lan, K. and Heidemann, J. 2006. A Measurement Study of Correlation of Internet Flow Characteristics. *Computer Networks 50,* 1, 46–62.

[74] Lautenschlaeger, W. and Feller, F. 2012. Light-Weight Traffic Parameter Estimation for On-Line Bandwidth Provisioning. In *Proceedings of the 24th International Teletraffic Congress*. ITC'12. 1–8.

[75] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. 1994. On the Self-Similar Nature of Ethernet Traffic. *IEEE/ACM Transactions on Networking 2,* 1, 1–15.

[76] Lucente, P. 2012. pmacct: a free open-source traffic accounting tool. `http://www.pmacct.net/lucente_pmacct_esnog10.pdf`. Online. Accessed Aug. 2014.

[77] Mai, J., Chuah, C.-N., Sridharan, A., Ye, T., and Zang, H. 2006. Is Sampled Data Sufficient for Anomaly Detection? In *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference*. IMC'06. 165–176.

[78] Makkonen, L. 2008. Bringing Closure to the Plotting Position Controversy. *Communications in Statistics – Theory and Methods 37,* 3, 460–467.

[79] Makkonen, L., Pajari, M., and Tikanmäki, M. 2013. Closure to "Problems in the extreme values analysis". *Elsevier Structural Safety 40*, 65–67.

[80] Mandjes, M. and van de Meent, R. 2009. Resource Dimensioning Through Buffer Sampling. *IEEE/ACM Transactions on Networking 17,* 5, 1631–1644.

[81] MAWI. Working Group Traffic Archive. `http://mawi.wide.ad.jp`. Online. Accessed Dec. 2013.

[82] McCloghrie, K. and Rose, M. T. 1991. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. RFC 1213.

[83] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM CCR 38,* 2, 69–74.

[84] Moshref, M., Yu, M., and Govindan, R. 2013. Resource/Accuracy Tradeoffs in Software-Defined Measurement. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. 73–78.

[85] Moshref, M., Yu, M., Govindan, R., and Vahdat, A. 2014. DREAM: Dynamic Resource Allocation for Software-defined Measurement. In *Proceedings of the ACM SIGCOMM*. 419–430.

[86] Norros, I. 1994. A storage model with self-similar input. *Queueing Systems 16,* 3–4, 387–396.

[87] Ntop. nprobe – an extensible netflow v5/v9/ipfix gpl probe for ipv4/v6. `http://www.ntop.org/products/nprobe/`. Online. Accessed Feb. 2014.

[88] Ntop. pf_ring – High-speed packet capture, filtering and analysis. `http://www.ntop.org/products/pf_ring/`. Online. Accessed Feb. 2014.

[89] Open Networking Foundation. 2009. OpenFlow Switch Specification – Version 1.0.0. `http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf`. Online. Accessed Jun. 2014.

[90] Open Networking Foundation. 2012. OpenFlow Switch Specification – Version 1.3.1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf. Online. Accessed Jun. 2014.

[91] Open Networking Foundation. 2013. OpenFlow Switch Specification – Version 1.4.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf. Online. Accessed Jun. 2014.

[92] Papagiannaki, K. 2003. Provisioning IP Backbone Networks Based on Measurements. Ph.D. thesis, University of London.

[93] Papagiannaki, K., Cruz, R., and Diot, C. 2003. Network Performance Monitoring at Small Time Scales. In *Proceedings of the ACM Internet Management Conference*. IMC'03. 295–300.

[94] Paxson, V. and Floyd, S. 1995. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking 3,* 3, 226–244.

[95] Phaal, P., Panchen, S., and McKee, N. 2001. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176.

[96] Pras, A., Nieuwenhuis, L., van de Meent, R., and Mandjes, M. 2009. Dimensioning Network Links: A New Look at Equivalent Bandwidth. *IEEE Network 23,* 2, 5–10.

[97] Puš, V. 2012. Hardware Acceleration for Measurements in 100 Gp/s Networks. In *Proceedings of the 6th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security*. AIMS'12. 46–49.

[98] QoSient. Argus. http://qosient.com/argus/. Online. Accessed Jun. 2014.

[99] Quittek, J., Zseby, T., Claise, B., and Zander, S. 2004. Requirements for IP Flow Information Export (IPFIX). RFC 3917.

[100] Sarvotham, S., Riedi, R., and Baraniuk, R. 2001. Connection-level Analysis and Modeling of Network Traffic. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. IMW'01. 99–103.

[101] Schönwälder, J. 2008. Simple Network Management Protocol (SNMP) Context EngineID Discovery. RFC 5343.

[102] sFlow.org. 2003. Traffic Monitoring using sFlow. http://www.sflow.org/sFlowOverview.pdf. Online. Accessed Mar. 2014.

[103] Sharafat, A. R., Das, S., Parulkar, G., and McKeown, N. 2011. MPLS-TE and MPLS VPNs with OpenFlow. In *Proceedings of the ACM SIGCOMM conference*. SIGCOMM'11. 452–453.

[104] SimpleWeb. Simpleweb. `http://www.simpleweb.org`. Online. Accessed Dec. 2013.

[105] SteinBerger, J., Schehlmann, L., Abt, S., and Baier, H. 2013. Anomaly Detection and Mitigation at Internet Scale: A Survey. In *Proceedings of the 7th International Conference on Autonomous Infrastructure, Management and Security*. AIMS'13. 49–60.

[106] The Simple Times. 2002. The quarterly newsletter of snmp technology, comment, and events – volume 10, number 1. `http://www.simple-times.org/pub/simple-times/pdf/vol10-num1.pdf`. Online. Accessed Jul. 2014.

[107] Tongaonkar, A., Keralapura, R., and Nucci, A. 2012. Challenges in Network Application Identification. In *Proceedings of the 5th USENIX Conference on Large-Scale Exploits and Emergent Threats*. LEET'12. 1–1.

[108] Tootoonchian, A., Ghobadi, M., and Ganjali, Y. 2010. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *11th PAM*. 201–210.

[109] van de Meent, R. 2006. Network Link Dimensioning: A Measurement & Modeling Based Approach. Ph.D. thesis, University of Twente. ISSN 1381-3617.

[110] van de Meent, R., Mandjes, M., and Pras, A. 2006. Gaussian Traffic Everywhere? In *Proceedings of the IEEE International Conference in Communications*. ICC'06. 573–578.

[111] van de Meent, R., Pras, A., Mandjes, M., van den Berg, H., and Nieuwenhuis, L. 2003. Traffic Measurements for Link Dimensioning, A Case of Study. In *Proceedings of the 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*. DSOM'03.

[112] van den Berg, H., Mandjes, M., van de Meent, R., Pras, A., Roijers, F., and Venemans, P. 2006. QoS-aware bandwidth provisioning for IP network links. *Elsevier Computer Networks 50,* 5, 631–647.

[113] van der Pol, R., Bredel, M., Barczyk, A., Overeinder, B., van Adrichem, N., and Kuipers, F. 2013. Experiences with MPTCP in an intercontinental OpenFlow network. In *Proceedings of the TERENA Networking Conference*. TNC'13. 1–8.

[114] Vilardi, R., Grieco, L. A., Barakat, C., and Boggia, G. 2013. Lightweight Enhanced Monitoring for High-Speed Networks. *Transactions on Emerging Telecommunications Technologies*, 1–19. Online. Accessed Dec. 2013.

[115] Yu, C., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G., and Madhyastha, H. V. 2013a. FlowSense: Monitoring Network Utilization with Zero Measurement Cost. In *14th PAM*. 31–41.

[116] Yu, M., Jose, L., and Miao, R. 2013b. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. 29–42.

[117] Yu, Y., Qian, C., and Li, X. 2014. Distributed Collaborative Monitoring in Software Defined Networks. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. 1–7.

[118] Zhang, Y. 2013. An Adaptive Flow Counting Method for Anomaly Detection in SDN. In *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 25–30.

[119] Zseby, T., Molina, M., Duffield, N., Niccolini, S., and Raspall, F. 2009. Sampling and Filtering Techniques for IP Packet Selection. RFC 5475.

Acronyms

AMX-IS Amsterdam Internet Exchange
Argus Audit Record Generation and Utilization System

CAIDA Center for Applied Internet Data Analysis
CDF Cumulative Distribution Function
CERN European Organization for Nuclear Research

HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure

IANA Internet Assigned Numbers Authority
IETF Internet Engineering Task Force
IP Internet Protocol
IPFIX IP Flow Information Export
IRTF Internet Research Task Force
ISP Internet Service Provider

MIB Management Information Base
MPLS Multiprotocol Label Switching
MRTG Multi Router Traffic Grapher

NNTP Network News Transfer Protocol

OVS Open vSwitch

Q-Q Quantile-quantile
QoE Quality of Experience
QoS Quality of Service

RRD Round-Robin Database

SDN Software-Defined Networking

SLA Service-Level Agreement

SNMP Simple Network Management Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

VLAN Virtual Local Area Network

VoIP Voice over IP

VPN Virtual Private Network

WIDE Widely Integrated Distributed Environment

YAF Yet Another Flowmeter

About the author

I was born in Passo Fundo, Rio Grande do Sul, Brazil, on April 2nd, 1985. I received my Bachelor of Science (B.Sc.) degree in Computer Science in 2007 from the University of Passo Fundo, Brazil, and my Master of Science (M.Sc.) degree in Computer Science in 2010 from the Federal University of Pernambuco, Brazil. In the period from Nov. 2010 to Nov. 2014 I was a Ph.D. candidate at the Design and Analysis of Communication Systems (DACS) group of the University of Twente, the Netherlands, under the supervision of Prof. Dr. ir. Aiko Pras and Prof. Dr. Hans van den Berg. During this period I had the opportunity to take part in four research projects, namely, EU FP7 UniverSelf, EU FP7 Flamingo NoE, EU FP7 MCN and SURFnet's Gigaport3 project for Next-Generation Networks. Below is a list of papers I published (or submitted) during the time I was a Ph.D. candidate at DACS, sorted in reverse chronological order:

- R. de O. Schmidt, R. Sadre, A. Sperotto and A. Pras, *Impact of Packet Sampling on Link Dimensioning*. Under review (IEEE TNSM).
- N. Bouten, R. de O. Schmidt, J. Famaey, S. Latré, A. Pras and F. De Turck, *QoE-Driven In-Network Optimization for Adaptive Video Streaming Based on Packet Sampling Measurements*. Under review (COMNET).
- R. de O. Schmidt, H. van den Berg and A. Pras, *Measurement-Based Network Link Dimensioning*. Under review (IFIP/IEEE IM 2015).
- R. de O. Schmidt, L. Hendriks, A. Pras and R. van der Pol, *OpenFlow-based Link Dimensioning*. Demo at Innovating the Network for Data-Intensive Science Workshop (INDIS), ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2014.
- R. de O. Schmidt, R. Sadre, A. Sperotto, H. van den Berg and A. Pras, *A Hybrid Procedure for Efficient Link Dimensioning*. Computer Networks (COMNET), 67, 252–269, 2014.
- F. Francois, N. Wang, K. Moessner, S. Georgoulas and R. de O. Schmidt, *Leveraging MPLS Backup Paths for Distributed Green Traffic Engineering*. IEEE Transactions on Network and Service Management (TNSM), 11(2), 235–249, 2014.

- R. de O. Schmidt, R. Sadre, N. Melnikov, J. Schönwälder and A. Pras, *Linking Network Usage Patterns to Traffic Gaussianity Fit*. In proceedings of the 13th IFIP Networking Conference, 2014.
- I. Drago, R. de O. Schmidt, R. Hofstede, A. Sperotto, M. Karimzadeh, B. R. Haverkort and A. Pras, *Networking for the Cloud: Challenges and Trends*. Praxis der Informationsverarbeitung und Kommunikation (PIK), Special Issue on Data Center Networking, 36(4), 1–8, Dec. 2013. (invited paper)
- R. de O. Schmidt, R. Sadre and A. Pras, *Gaussian Traffic Revisited*. In proceedings of the 12th IFIP Networking Conference, 2013.
- R. de O. Schmidt, R. Sadre, A. Sperotto and A. Pras, *Lightweight Link Dimensioning using sFlow Sampling*. In proceedings of the 9th International Conference on Network and Services Management (CNSM), 152–155, 2013.
- M. Hoogesteger, R. de O. Schmidt, A. Sperotto and A. Pras, *Reports on Internet Traffic Statistics*. Student Poster at TERENA Networking Conference (TNC), 2013.
- R. de O. Schmidt, A. Sperotto, R. Sadre and A. Pras, *Estimating Bandwidth Requirements using Flow-level Measurements*. Student Poster at TERENA Networking Conference (TNC), 2012.
- R. de O. Schmidt, A. Sperotto, R. Sadre and A. Pras, *Towards Bandwidth Estimation using Flow-level Measurements*. In proceedings of the 6th International Conference on Autonomous Infrastructure, Management and Security (AIMS), LNCS 7279, 127–138, 2012.
- R. de O. Schmidt and A. Pras, *Estimating bandwidth requirements using flow-level measurements*. In proceedings of the 5th International Conference on Autonomous Infrastructure, Management and Security (AIMS), LNCS 6734, 169–172, 2011.
- R. de O. Schmidt, A. Pras and R. Gomes, *On the Evaluation of Self-Addressing Strategies for Ad-Hoc Networks*. In proceedings of the 17th International Workshop EUNICE, LNCS 6955, 31–42, 2011.
- R. de O. Schmidt, R. Gomes and A. Pras, *Evaluating Self-Addressing Protocols for Ad-Hoc Networks*. Student Poster at NWO - ASCI - ICT.OPEN, 2011.
- R. Gomes and R. de O. Schmidt, *Evaluating automatic pools distribution techniques to self-configured networks*. In proceedings of the 16th IEEE Symposium on Computers and Communications (ISCC), 658–663, 2011.
- A. Pras, A. Sperotto, G. C. Moreira Moura, I. Drago, R. R. R. Barbosa, R. Sadre and R. de O. Schmidt, *Attacks by "Anonymous" WikiLeaks Proponents not Anonymous*. Technical Report TR-CTIT-10-41, University of Twente, 2010.