Flávio Dinis Gonçalves Rosa Jacinto

Master of Science

Internet Measurement

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Luís Bernardo, Professor,
Universidade NOVA de Lisboa

Júri

Presidente: Luís Augusto Bica Gomes de Oliveira
Arguentes: Pedro Miguel Figueiredo Amaral
Vogais:

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2017

Internet Measurement

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

To the ones that fulfill my heart,

Acknowledgements

Por onde começar? Dizem que somos o que fazemos, por isso em última instância podemos afirmar que somos o que as nossas acções espelham. As pessoas citadas neste documento, são aquelas cujas acções tiveram em mim o impacto de lhes agradecer aqui! Assim, partindo de uma escala ampliada, agradeço a Deus por me fazer acreditar. Acreditar que é possível ter pessoas como os meus pais, Dinis e Helena Jacinto, e avós, Manuel e Elvira Gonçalves, Francisco Jacinto e Maria Rosa, que foram incansáveis no apoio que deram em toda a vida, suportando sempre os meus objectivos e ambições. A eles dedico o meu maior agradecimento, pois sem eles nada disto seria possível. Mas porque não há impossíveis, agradeço com um carinho especial à Margarida Freixo, menina dotada e sensata pela qual me apaixono todos os dias. Acreditar que é possível ter amigos como, o Fábio Cruz, a quem dedico um especial agradecimento pela paciência e compreensão que demonstrou para comigo ao longo desta jornada em toda em qualquer matéria, o Rodrigo Santos a quem agradeço pelo desenho do logótipo da aplicação Android, ou o Guilherme Góis que me ajudou em questões da linguagem Java e diversas questões relacionadas. Não parando de acreditar, digo Obrigado, ao Jorge Dias pelo companheirismo, à Raquel pelo anhanço, ao João Baptista pelo avacalho, à Rafa pela sensatez, à Nádia pelo compromisso, ao Diogo pela tranquilidade, ao Ricardo Ferreira pela extravagância, à Ana Lúcia pela espírito de criança, ao Rui Baptista pela animação, ao Gabriel Anacleto pelo acolhimento, ao Miguel Santos pela coragem, ao Pedro Nunes pela responsabilidade, à Sara Martins pela determinação, ao Bernardo Melo pelos conselhos sábios, ao Micael Fernandes pelo detalhe, ao Francisco Silveira pela ambição, ao Guilherme Carvalheiro pela mudança, ao João Eusébio pela humildade e ao Tiago Pina pela apadrinhamento. Por último, mas não menos importante, quero agradecer ao meu orientador, o Professor Luís Bernardo pelo rigor e profissionalismo dedicados, ao agrupamento 895 São João da Talha bem como ao escutismo em geral pela formação enquanto jovem e adulto, e claro, à gloriosa FCT, por ser ontem, hoje e amanhã a casa de aqueles que por aqui passam. Assim concluo, acreditando que é possível ser melhor e chegar mais longe, se ouvirmos os ensinamentos daqueles que cruzam a nossa vida, guardando para nós aqueles que nos fazem evoluir enquanto pessoa, por isso o meu mais sincero obrigado a todos aqueles que contribuiram para a realização deste sonho. Quero ainda agradecer à Fundação para a Ciência e a Tecnologia (FCT) pelo suporte dado pela ao projeto UID/EEA/50008/2013 (Instituto de Telecomunicações), que financiou os projetos internos VELOCE-MTC e LoCoSDN."

Abstract

Nowadays, TCP channel estimation is a matter of great importance, being communication network metrology the core of network performance analysis field, since it allows to interpret and understand the network behaviour through the gathered metrics. In the context of this dissertation, an open source software project, available on GitHub, was developed. It uses a client-server architecture to estimate the Bulk Transfer Capacity (BTC) and provides portability due to Java and Android clients, being able to run on computers, tablets and mobile phones.

Two algorithms to measure the BTC were deployed. Their measuring capacity was analysed and optimized, supported on studies about the influence of the TCP windows. The packet train dispersion algorithm was also implemented and analysed, but it did not allow measuring significant BTC results. The performance of the tool was tested for wired and cellular wireless networks, considering all the major Portuguese network operators. The results were compared to the ones measured by the iPerf3 reference tool, considering a stop criteria based on Jain's Fairness Index [1] in order to inject the less possible traffic into the network.

The measurement results are in line with the methodology proposed by ETSI and Ofcom to monitor the bandwidth, considering fixed time transmissions, and can contribute to reduce the transmission durations required to analyse each network.

Keywords: metrology, BTC, packet train dispersion, Ofcom, iPerf3, ETSI

Resumo

Actualmente, a estimação da capacidade de uma rede é uma questão de grande importância, estando a metrologia de redes de comunicação no centro do campo da análise do desempenho de redes, uma vez que permite interpretar e compreender o comportamento da mesma através das métricas obtidas. No contexto desta dissertação, foi desenvolvido um projecto cujo código público está disponível publicamente no GitHub. Este usa uma arquitectura cliente-servidor para estimar o *Bulk Transfer Capacity (BTC)*, garantindo portabilidade devido aos clientes Java e Android, possibilitando a execução em computadores, tablets e telemóveis.

Foram implementados dois algoritmos para medir o BTC. A capacidade de medição da ferramenta foi analisada e optimizada através de estudos sobre a influência das janelas do TCP no débito. O algoritmo *packet train dispersion* também foi implementado e analisado, mas não permitiu medir resultados significativos de BTC. O desempenho da ferramenta foi testado em redes com e sem fios, incluindo as celulares, considerando todos os principais operadores de rede móvel Portugueses. Os resultados foram comparados com os medidos pela ferramenta iPerf3, que foi usada como referência, considerando um critério de paragem baseado no *Jain's Fairness Index* de modo a injectar na rede o menor tráfego possível.

Os resultados das medições seguem a metodologia proposta pelo ETSI e pela Ofcom para monitorizar a largura de banda considerando as transmissões de tempo fixo, podendo contribuir para reduzir a duração das transmissões por cada rede analisada.

Palavras-chave: metrologia, BTC, *packet train dispersio*n, Ofcom, iPerf3, ETSI

Contents

List of Figures

List of Tables

Acronyms

ACK Acknowledgement.

ADR Asymptotic Dispersion Range.

ADSL Asymmetric Digital Subscriber Line.

AOSP Android Opensource Smartphone Platform.

APK Android Application Package.

ART Android Runtime.

BART Bandwidth Available in Real-Time.

BTC Bulk Transfer Capacity.

DNS Domain Name Space.

DQR Disjoint Queuing Region.

ESnet Energy Sciences Network.

FTP File Transfer Protocol.

GPON Gigabit Passive Optical Network.

GPS Global Positioning System.

GUI Graphical User Interface.

HAL Hardware Abstraction Layer.

ID Identification.

IGI Initial Gap Increasing.

IP Internet Protocol.

IPPM IP Performance Metric.

IRB Institutional Review Board.

ISP Internet Service Provider.

JQR Joint Queuing Region.

JSON JavaScript Object Notation.

MIB Management Information Base.

MITATE Mobile Internet Testbed for Application Traffic Experimentation.

MRTG Multi Router Traffic Grapher.

MSU Montana State University.

MTU Maximum Transfer Unit.

MVC Model-View-Controller.

NAT Network Address Translation.

OS Operating System.

OWAMP One-Way Active Measurement Protocol.

PerfSONAR Performance Serive Oriented Network Monitoring Architecture.

PGM Probe Gap Model.

PMTU Path Maximum Transmission Unit.

PRM Probe Rate Model.

PTD Packet Train Dispersion.

QoS Quality of Service.

RNP Rede Nacional de Pesquisa.

RTT Round Trip Time.

SLA Service Level Agreement.

SLoPS Self-loading periodic streams.

SNMP Simple Network Management Protocol.

TLS Transport Layer Security.

TTL Time to live.

UDP User Datagram Protocol.

UI User Interface.

UK United Kingdom.

UML Unified Modelling Language.

URL Uniform Resource Locator.

VoIP Voice over IP.

XML Extensible Markup Language.

Chapter 1

Introduction

1.1 Motivation

Internet measurement has been a relevant field in the comprehension and analysis of the network performance. Communication network metrology, which is defined as the science of network measurement, plays an important role in interpreting results through the metrics related with IP networks. These metrics can be obtained actively by injecting probe packets into the network, and passively by simply gathering the IP packets headers. In one way or another, measurement studies give to researchers, developers, regulators and the general public an overview of the bandwidth available and how much a TCP connection is able to use it. There are several techniques that perform communication network measurement in order to estimate the maximum achievable throughput of a network without causing congestion. Since the most problematic issue in TCP is to distinguish whether the packet was lost by congestion or due to the effects of the wireless channels, estimation became very important because it provides a mechanism to improve the TCP performance, and the detection of the network capacity limits.

Internet measurement is the key to verify whether the contract requirements of the Internet Service Provider (ISP) are being met. These requirements are established in Service Level Agreement (SLA), which are contracts where the agreed terms between the ISP and the costumer are defined. In order to accomplish the purposed intent, several tools have been deployed in the past few years, including in cellular networks field, where due to cell coverage limitations, network performance falls. Accordingly, in the context of this thesis, Java and Android applications were developed and used to estimate the Wi-Fi and cellular networks throughput. An alternative method based in the packet train dispersion algorithm [2] was also tested, but it did not prove to be able to provide reliable measurements.

1.2 Contributions

The main goal of this dissertation is centered on the development of a Java measurement tool, that performs Bulk Transfer Capacity (BTC) estimation in line with the UK communication regulator (Ofcom) standard. Thus, BTC which is defined as the end-to-end throughput achievable using the TCP protocol, is measured by the developed tool using Java as an higher cross functionality programming language. According to this, a client-server architecture was built, and a new stop criteria was proposed, which reduces the traffic injected into the network, without affecting the measurements accuracy.

1.3 Document Structure

The document starts with this chapter, where the motivation, as well as the contributions for the dissertation are presented. In the second chapter, the related state of the art is presented, including the algorithms and tools used to measure the throughput of a network followed by the client-server architecture in the third chapter, where the deployment and operation of both end points are covered in detail. The measurement results are presented and explained in chapter 4, being the document finished with the conclusions and future work in chapter 5.

CHAPTER 2

STATE OF THE ART

2.1 Basic concepts

Metrology is defined as the science of measurement, which includes all related theoretical and practical aspects. A new research area described as communication network metrology has been evolving and is being used to improve the congestion control scheme of TCP. Bandwidth related metrics such as capacity, available bandwidth and Bulk Transfer Capacity (BTC) are introduced in this section.

2.1.1 Capacity

Capacity is a measurement defined as the maximum amount of data that may be transferred between two end points of a network. Hence, it is possible to distinguish between links at data link layer and at the Internet Protocol (IP) layer. In both cases the term rate comes up, since capacity is an upper bound for transfer rate for both mentioned layers.

Moreover, it is possible to express the layer 2 transmission rate delivered to the IP layer as a function of packet size. Prasad *et al* [3] define capacity as the maximum bit rate of a hop measured at the network layer, at which the hop can transfer Maximum Transfer Unit (MTU)-sized packets. Furthermore in the text, C_i denotes the capacity of hop i. The minimum link capacity of a path is the metric that determines the end-to-end capacity of a path with H hops, which is expressed by,

$$C = \min_{1,\ldots,H} C_i \tag{2.1}$$

2.1.2 Available Bandwidth

The capacity of a link is restricted by the underlying propagation medium. Hence, the available bandwidth additionally depends on the traffic load that crosses the link, which usually varies with time.

Thus, the available bandwidth of a link is the spared capacity of that link during a certain period of time. Extending the previous definition in terms of average metrics, it is possible to define u_i as the average utilization of hop i in a given time interval. This leads to the definition of available bandwidth, which can be expressed as the unutilized fraction of capacity in a given hop i as

$$A_i = (1 - u_i)C_i. \tag{2.2}$$

Similarly to the capacity case, the minimum available bandwidth of hop i in a given path determines the end-to-end available bandwidth.

2.1.3 Bulk Transfer Capacity

The bulk transfer capacity is the maximum throughput achievable by a single TCP connection. This metric can be calculated as the average number of bytes transmitted per time unit [4], using the TCP protocol. In contrast to available bandwidth, which assumes that average traffic load remains constant, the BTC depends on how the bandwidth is shared with concurrent TCP flows.

2.1.4 One-way delay

The one-way delay is defined in [5] as the time a packet takes from source to the destination, which is often calculated as the half of Round Trip Time (RTT). However, as Paxson mentioned in [6], paths tend to be more asymmetric, which means gradually the links bandwidth measurement will be different in both directions. Actually, this is what happens nowadays, since the upload speed is lower than download, and this contributes to preserve the devices battery.

IP Performance Metric (IPPM) is defined in RFC 2679 [7] as the one-way delay the first bit of a packet takes from the source at time T, to the receiver, at time $T + dT$, i.e. dT.

The measurement process is executed by calculating the delay. Since the packet is stamped at time T on the source, the destination just needs to read out the clock in order to obtain the delay. Obviously, this kind of technique implies synchronization in both end-systems, which may lead to measurement errors. Hence, the calibration has the important role of determining systematic and random errors generated by the instruments, since generally, the measured value is the sum of true value, systematic error and random error. The value measured also depends of the protocol (User Datagram Protocol (UDP) or TCP), size, etc., which define different contents to the IP packet [7].

One-way delay measurement is deployed in One-Way Active Measurement Protocol

(OWAMP) [8], allowing the measurement in the forward and reverse path according to IPPM recommendations.

2.1.4.1 One-way delay variation

Delay variation, often called jitter, is a key metric to ensure quality of service in many applications.

The IP packet delay variation or delay variation can be calculated from a selected pair of packets within a stream that is being transferred between two measurements points. Hence, the difference between the one-way delay of the selected pair of packets is also called delay variation.

The measurement is done by computing the delay difference between two consecutive packets, which does not need clock synchronization since errors will cancel each other when calculation occurs. Related with uncertainties in clocks, the resolution problem appears, because at end-systems the clock precision may be compromised. For example, in Linux operating systems older than version 2.2.0, the clock resolution is about $10ms$ [5].

2.1.5 Round-trip time

The RTT is the delay that a data packet takes to travel from the source to the destination plus the delay required for an Acknowledgement (ACK) to travel back to the source.

The measurement of RTT is simpler than one-way delay because the calculation of transmission times is only done at the source. When the source sends the first bit of a packet, it stamps the current time in the packet. At the receiver side, when the packet is fully received, it sends an acknowledgement response to indicate the successfully delivery. Finally, the RTT is calculated at the source through the difference between the time stamp and the arrival time of the ACK.

Ping is an example of non-cooperative tool that uses ICMP echo packets to measure the time taken from the source to the destination plus the time taken by the ICMP reply packets to come back.

2.1.6 Packet loss

TCP uses packet loss as a signal of congestion. The loss pattern or loss distribution is a key parameter based on statistical association of loss distance and loss period that is crucial in certain applications like audio or video. As defined in RFC 3357 [9] by IETF, the loss distance is the difference between the *"sequence numbers of two successively lost packets which may or may not be separated by successfully received packets."* A loss period metric measures the frequency and length of consecutive lost packets. According to this, loss distance could gather the spacing between loss periods.

Additionally, average metrics such as the average length of loss periods and the

average inter-loss periods can also be calculated. Another statistical process called one-way loss noticeable rate defines that a packet loss is noticeable when the distance between two consecutive lost packets is lower than a given Δ, defined as loss constraint. Figure 2.1 illustrates an example of the referred technique, where a sequence of packets is sent, being the encircled packets classified as noticeable losses, since the loss constraint is lower than a defined value, $\Delta = 99$ in this case [5].

One-way packet loss measurement is deployed in OWAMP allowing the measure to occur in forward and reverse channels. It is through the time-out values that detection of non-received packets is performed. If the ACK corresponding to the sent data packet is not received in a defined period of time, the packet is considered lost. Moreover, a technique based on TCP acknowledgements, performed in two phases, may also be applied. The data-seeding phase is the first phase and it consists on sending at the source a series of TCP packets. The second phase, called hole-filling phase, has the role of discovering lost packets sent in the first phase. The source assumes that when the last acknowledgement corresponds to the last sent packet, no packet has been lost. Otherwise, the source will record and retransmit the lost packets until all of them are acknowledged.

Since in most of the TCP implementations receivers wait approximately 100 to 500ms to respond, they do not respond with one ACK per packet. This raises the demand for forcing the destination to send those acknowledgements during the tests. One way to do that is sending packets out of order to trigger the fast-retransmission algorithm. This algorithm also known as duplicate ACKs method, do retransmissions if three repeated ACKs are received. Therefore, it is possible to calculate the loss rate at the source on the reverse path by knowing the sequence numbers of the sent packets and the sequence numbers of the acknowledges.

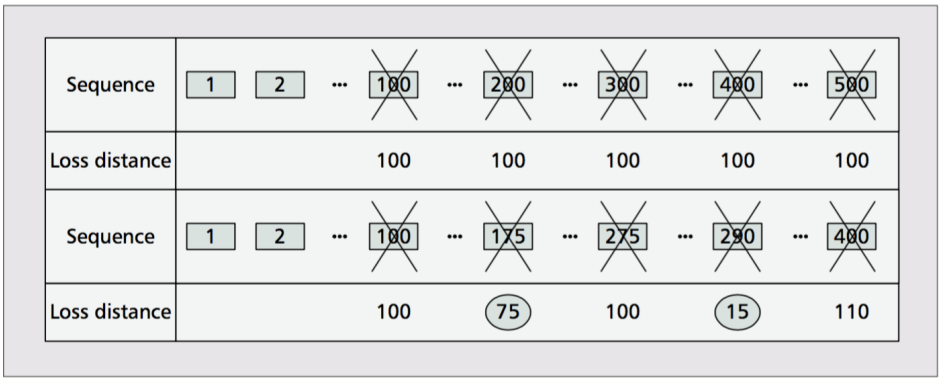

Figure 2.1: Noticeable losses indicated by encircled numbers with $\Delta = 99$ [5]

2.1.7 Packet reordering

TCP is the most important management protocol of transport layer, providing a reliable full-duplex channel for end-to-end data communication. It is a connection oriented protocol which performs error control as well as congestion control. According to this,

out of order packet delivery may lead to unnecessary retransmissions and/or unnecessary reduction of the congestion window, which lead to an inefficiency of the referred protocol.

Ordered delivery is crucial specially for real-time media applications. Multi-path routing, parallelism in networks devices, layer 2 retransmissions and multiple buffers with different rate services are reasons that may change packet delivery order. Hence, an effort has been made in order to enhance the retransmission scheme of TCP. Consequently, OWAMP protocol allows the measurement to be performed along the forward and reverse path by sending and analyzing streams of packets. Therefore, a new set of techniques presented in [10] provides a reliable one-way reordering estimation in both directions between a probe host and most TCP-based servers on the Internet. Those tools implement techniques which perform tests by establishing one or more connections to a remote host in order to verify whether the packets are correctly delivered.

The single connection test uses the standard three-way handshake for establishing the connection. Once this connection is established, each sample is measured in two different phases: a preparation phase and a measurement phase. Since most of TCP implementations use a delayed acknowledgment algorithm in the hopes of combining a data frame with an ACK on an outgoing packet, it forces the source to send a packet with an unexpected sequence number in order to receive an ACK for each sent packet as illustrated in figure 2.2. This feature allows the sender to infer if both packets and ACKs were reordered in flight. As shown in figure 2.2, the preparation phase occurs by sending an out of order packet (data 2) to the remote host. On the second phase the reordering measurement is performed by sending two sample packets. If the packets are delivered out of order the ACK 1 will be received followed by ACK 4, meaning that data packets were not reordered in flight since data 3 arrived first than data 1. Otherwise, if the ACK 4 arrives at the sender first, it will confirm the whole series meaning that packets were reordered in flight. The same principle can be applied to the reverse channel as figure 2.2 demonstrates, whereas data 3 was delivered after data 1, however, ACK 4 was received before ACK 3, meaning that the acknowledgements were also reordered in flight.

Although this approach may perform a reliable packet reordering measurement, it has some limitations due to the fact that it requires two data packets to evaluate whether the reordering occurred or not, since if one of the data packets or ACKs were lost, there is no possible evaluation. Another coincident problem related with the remaining sample packet is that when this packet has been discarded (*e.g.* situations with considerable loss rates) that may influence the measurements if the loss is correlated with reordering. The dual connection test was deployed in order to deal with the previous limitations. In this approach, two sample packets with sequence numbers greater than expected are sent to the remote host by two different connections, being one sent for each connection. The pair of packets are labeled in the identification field (IPID) of the IP packet header with an unique value which makes possible to determine both, the order in which the remote host sent acknowledgement packets (reverse path reordering) and the order in which sample packets were received in the remote host (forward path reordering). This may

be performed by comparing the difference between the IPIDs of the acknowledgments and the order in which the sample data packets were sent. Since two connections are used, it is always possible to associate each data packet with source and destination port numbers. Notice that in the majority of systems, the Nagle's algorithm [11] is enabled by default, causing a delay in the acknowledgments in order to improve the efficiency of TCP/IP networks.

Altough this approach mostly eliminates this delayed algorithm problem by using two different connections, this can be problematic when the remote host uses a load balancer. Besides that, this approach assumes that IPID uses an increasing function which may not always be true.

Since the load balancers always forward packets of a given TCP connection to the same host, a third method has been proposed to solve this issue. The SYN test method takes advantage of TCP three way handshake by sending a pair of SYNs with different starting sequence numbers. As illustrated in figure 2.3, the sender will receive different sequences of packets depending whether the reordering happened on the forward or on the reverse path. When the first SYN arrives it will trigger the SYN RECV state of the remote host, which will generate a SYN/ACK response. On the other hand, when the second SYN arrives, the sequence number is verified, and when is inside the window range a RST response should be generated. Otherwise the remote host will answer with a pure ACK to indicate that the SYN arrived out of order.

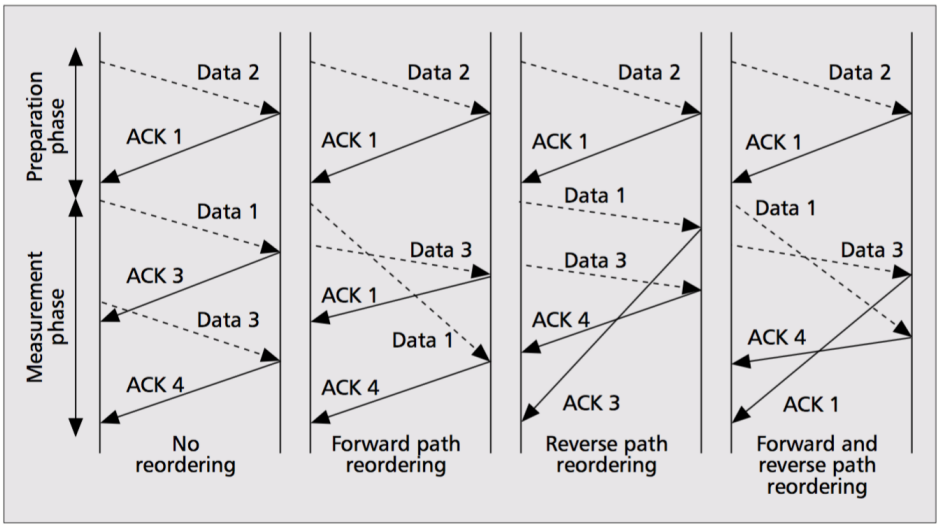

Figure 2.2: Sequences of 1 byte TCP data packets generated by the single connection test. All forward path packets are denoted with a dashed line and all reverse path packets a solid line [5]

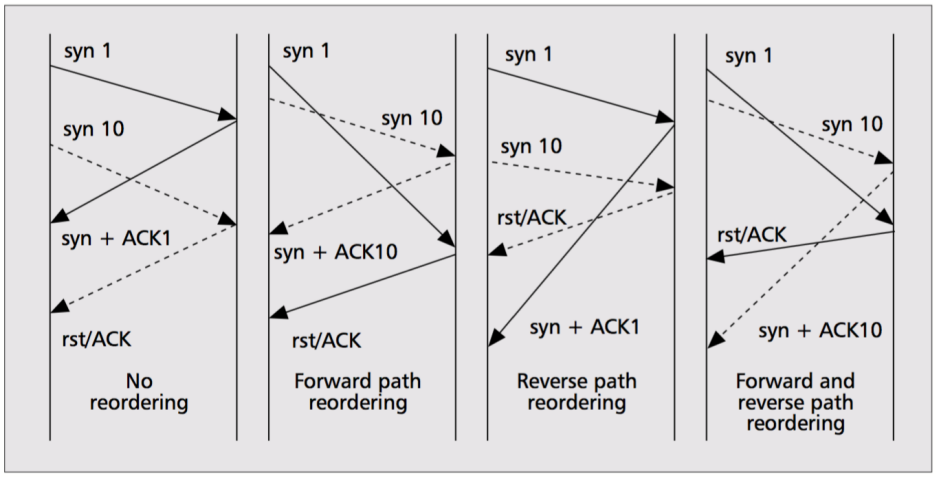

Figure 2.3: SYN test [5]

2.1.8 Route

In a network, each node is identified by its IP address. A route is an ordered sequence of nodes that represent a path between two end points of a network. A methodology to discover the route has been deployed in traceroute, which is a non-cooperative tool. This tool takes advantage of Time to live (TTL) field which is decremented by one unit for each router on the path. The trace of the route is performed by first sending User Datagram Protocol (UDP) packets from the source to the destination with TTL equal to one. When the first router of the path receives the packet, it sends an ICMP message to the source indicating that the packet was discarded. Those ICMP messages contain the router address in the headers, which allows the source to identify that router as the first hop of the path. The same principle is applied in the next iterations of the algorithm by just raising the TTL value, the source should be able to identify every hop on the path. When the destination sends an ICMP message the technique is completed.

However, there is no guarantee that probe packets coming from different hops will follow the same route as previous probes. Even when the route does not change at the IP layer, successive IP routers can be connected through different layer 2 technologies. Another problem is related with the fact that the sender only identifies the hops on the path through the received ICMP messages, which might be a problem since not all routers send them.

2.1.9 Bandwidth

In the context of the transport or application layers of the TCP/IP model, the term bandwidth defines bit rate according to most of the authors. Hence, bandwidth is the amount of data that can be transferred per unit of time, which along a digital network is the same as throughput.

In the first section of this state of the art, some basic concepts related with measurement metrics were introduced, such as capacity, available bandwidth and bulk-transfer capacity. In the following sections these bandwidth related metrics will be presented along with the techniques and tools used in the context of TCP/IP networks to measure them.

2.1.9.1 Available bandwidth measurement

As seen in the basic concepts section, the available bandwidth of a link is the free capacity of the link during a certain period of time. The techniques of available bandwidth measurement can be divided into three categories defined as Packet Train Dispersion (PTD) [2], Probe Gap Model (PGM) [12] and Probe Rate Model (PRM) [5]. Moreover, a program called Multi Router Traffic Grapher (MRTG) [13] will also be introduced, since it is a reference in this matter.

2.2 Algorithms

Network metrology has been widely studied on many research projects and laboratories. There are several features related with traffic which are important in Internet measurement such as characterization, volume, nature, modelling and matrices. Network cartography and network mapping, which consists on the research of physical connectivity of networks, have been areas of interest as well, since networks have become more and more dynamic and complex.

Moreover, Internet Service Provider (ISP) negotiate Quality of Service (QoS) levels through a standardized service contract named SLA on which metrology can be useful to verify whether the established requirements are met.

2.2.1 Active Measurements

Active measurements are a set of techniques for network estimation that consists on sending probe packets with particular properties (packet size, inter-departure time, bitrate, etc.) from the source to the destination. Deployed algorithms that perform this kind of measurement are presented in this section. Tools that implement the following algorithms may be classified by their intrusiveness according to Prasad *et al.* in [1] who claims that "*an active measurement tool is intrusive when its average probing traffic during the measurement process is significant compared to the available bandwidth in the path.*" In fact, the use of active measurement tools results in intrusiveness problems on the network traffic, being an example of that the tools that force routers and hosts to generate ICMP packets. For this reason, and in association with security purposes, there has been the necessity of filtering incoming Internet Control Message Protocol (ICMP) traffic, which results in disrupted measurements.

Tools that perform active measurements can be classified into cooperative and non-cooperative. Cooperative or passive measurement tools consist on computing metrics such as latency, rate and inter-packet gap by analyzing probe packets at the receiver side, which requires separated pre-installed software on both source and destination. Otherwise, the non-cooperative tools obtain in a indirect way all statistical information on the probes.

There are timing considerations such as clock accuracy, clock stability and clock synchronization, which can be a constraint in the measurement of end-to-end systems, as will be seen in the following algorithms. Passive measurement concepts will be mentioned along the text, reason why they will be introduced now as the techniques that are carried out by observing network traffic flows through the analysis of captured packet headers.

2.2.1.1 Packet train dispersion

The packet train dispersion is a technique implemented by cprobe [2], which consists on sending N ICMP echo request packets of size L to a destination in order to get ICMP reply packets in response, where $N > 2$. The dispersion denoted by $\Delta(N)$ is the amount of time between the reception of the first packet and the reception of the last packet. The available bandwidth, which is the same as throughput, is obtained by dividing the number of bytes sent by the elapsed time between the first and the last bits received as denoted by the equation 2.3 [5]. A stream of packets is sent with the objective of raising the probability of competing traffic load or cross traffic between the train probe packets. If the packets from the probe tool are the only that are queued at the bottleneck link, the destination is able to measure the bottleneck bandwidth based on the inter-arrival time of those packets. Since the primary role of this method is the measurement of the available bandwidth, if the sending rate of probe packets is lower than the bottleneck speed, the absence of cross traffic will be felt. In this scenario, a metric called Asymptotic Dispersion Range (ADR), which relates to the utilization of all links in the path, is equal to the end-to-end capacity. In all other cases (i.e. with concurrent load), the ADR is not related with the available bandwidth and if the length of packet train is increased the measured variance is reduced, but the estimates converge to an ADR value which is lower than the capacity [14]. Asymptotic Dispersion Range may be a useful metric for monitoring the quality of service that the path offers, being the approach deployed in pathrate [14].

$$A = \frac{(N-1)L}{\Delta(N)} \tag{2.3}$$

2.2.1.2 Self-loading periodic streams (SLoPS)

SLoPS [15] is a technique that performs end-to-end available bandwidth measurements. This method consists on sending a periodic packet stream of equal sized packets at rate R from the source to the sink in order to monitor one way delay variations of the sent packets. If R is greater than the available bandwidth A, the delay will increase because packets are being queued in the path. Otherwise, if $R < A$ no congestion occurs and the delay will remain constant. Moreover, the packet streams are spaced at the sender by creating a silent period between each stream, which maintains the average probing traffic rate below ten percent of the available bandwidth.

Due to its iterative nature, this algorithm may have a long convergence time in obtaining the available bandwidth value.

2.2.1.3 Probe rate model

The PRM model's primary objective is the detection of the point on which the delay along the path increases, because at this point the sending rate of the probing packets is equal to available bandwidth. If the source sends probe traffic with a rate lower than the available bandwidth, the rate will be the same along the path because no packets are queued. On the other hand, if the sending rate raises, this technique will be able to seek for the turning point where the sending rate matches the arrival rate. This turning point is the available bandwidth measurement.

The probe rate model is implemented by two cooperative tools: Pathload [16] and PathChirp [17]. The Pathload implements an algorithm based on SLoPS technique which was explained above. On the other hand, PathChirp concerns are reducing the probing traffic that is loaded on the network. It performs that by sending a chirp probing train on which the probe packets are exponentially spaced. Imagine a train chirp of n packets. This train only needs $2n-2$ packets using a packet pair technique to exploit $n-1$ packet spacings. Additionally a single chirp train is able to perform all the network probing at different rates.

2.2.1.4 Probe gap model

The probe gap model (PGM) is a method which was initially implemented in cooperative tools, namely Initial Gap Increasing (IGI) [12] and Spruce [13]. The PGM uses a packet pair to evaluate the available bandwidth by analyzing the pair dispersion. As is illustrated in the figure 2.4, two successive packets are sent with a time gap Δ_{in}. Those packets arrive at the destination with a time gap Δ_{out}, which means that the other traffic that crosses the bottleneck is transmitted in $\Delta_{out} - \Delta_{in}$. According to this, the cross traffic rate is given by $\frac{\Delta_{out} - \Delta_{in}}{\Delta_{in}} C$ where C is the capacity of the bottleneck. Hence, the available bandwidth of

the link is given by

$$A = \left(1 - \frac{\Delta_{out} - \Delta_{in}}{\Delta_{in}}\right) C \tag{2.4}$$

On the other hand, IGI has a different approach: it defines two regions with different queueing periods. A queueing period is defined as the time gap on which the queue is not empty. Hence, between two consecutive queueing periods the queue is empty. Two different regions are defined: Disjoint Queuing Region (DQR) and Joint Queuing Region (JQR). The Initial Gap Increasing technique operates under the JQR condition if the queue of the bottleneck does not become empty between the time of the first packet leaves the router and the second arrives, otherwise it operates under DQR. This is guaranteed if Δ_{in} is smaller or equal to the transmission time of the probe packet. However, if Δ_{in} is too small, it could flood the bottleneck. The experimental results carried out in [12] demonstrate that IGI tool starts with a small Δ_{in} that is increased until the optimal point, which occurs when the average output gap is equal to the initial gap. Thus, IGI determines the available bandwidth assuming that the cross traffic C_T at the bottleneck is given by $C_T = \frac{C\Delta_{out} - L}{\Delta_{in}}$, where L is the size of the probe packets and C is the end-to-end capacity. Note that this approach defines $\Delta_{out} = \frac{L + C_T \Delta_{in}}{C}$. Furthermore, the authors demonstrate that packet-pair dispersion increases linearly with the cross-traffic rate if the JQR condition is valid.

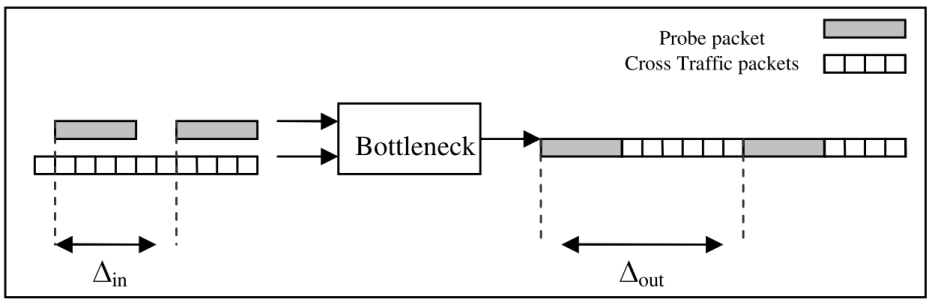

Figure 2.4: Probe gap model [13]

2.2.1.5 Multi Router Traffic Grapher (MRTG)

The MRTG is a tool that uses Simple Network Management Protocol (SNMP). SNMP is a protocol that performs the network management in an IP network by collecting and analyzing the information about managed devices such as routers, switches, servers, hubs, printers on which each device has an associated Management Information Base (MIB). MRTG reads the load of the routers by collecting the information present in the MIB of each router.

Figure 2.5 shows the available bandwidth measurements performed over a period of 24 hours, comparing Pathload, IGI, Spruce and MRTG. During the period from hour 5 to 10, the cross traffic was injected at a rate of 20 Mb/s. From hour 10 to 12, the generated

cross traffic rate increased to 40 Mb/s, being absent the rest of the time, where only the monitoring traffic was present in the path. Spruce demonstrates the best performance among approaches of figure 2.5.

Multi router traffic grapher simulates traffic bitrate by generating HTML pages with images with a measurement granularity no less than five minutes. A newer version, called MRTG++, has improved its granularity to ten seconds using an Autonomous System (AS) tool know as Available Bandwidth Estimator (ABEst), which provides information about link capacity and utilization. Since ABEst uses prediction to infer the available bandwidth, it can reduce the signalling and router load. Otherwise, an inherent limitation relates to the fact that it requires routers to modify the probing packets with data originated from the MIB.

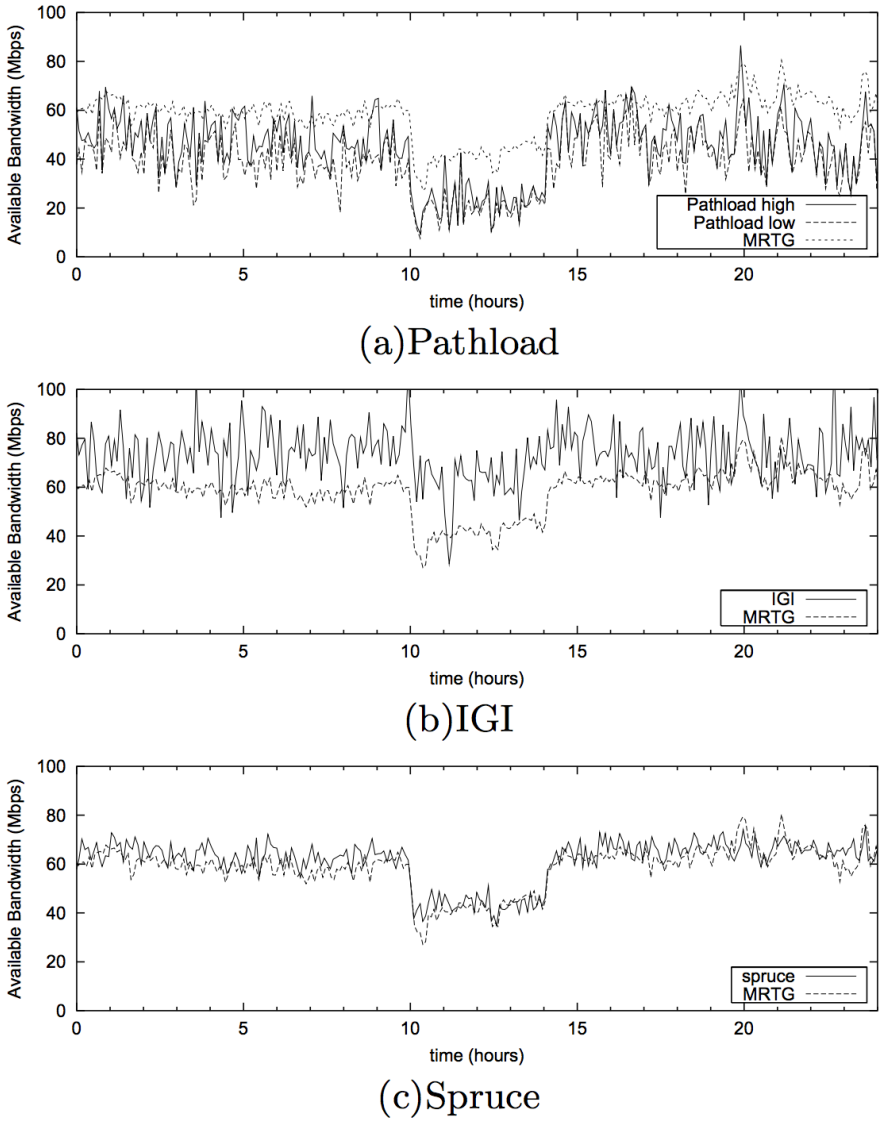

Figure 2.5: Available bandwidth estimation comparison between Pathload, IGI, and Spruce with 100 Mb/s path connecting UC Berkeley to MIT Laboratory for Computer Science (LCS) [13]

2.2.1.6 End-to-end capacity

End-to-end capacity concept has been seen in the first section of this state of the art as being the minimum value of the capacities among all hops that composes the path. Despite that, it is possible to measure the end-to-end capacity directly by applying the packet pair dispersion technique, which is presented in this section.

Packet pair dispersion

The packet pair technique consists on sending two consecutive packets large enough to queue them at the bottleneck link. This method assumes that the bottleneck router uses FIFO queueing and also assumes that the pair of packets are sent with no other cross traffic between them. This pair arrives at the link i with a Δ_{in} time distance being L the size of both packets. As shown in figure 2.6, the packet pair after crossing the bottleneck has a time dispersion equal to $\Delta_{out} = max\left(\Delta_{in}, \frac{L}{C_i}\right)$, where C_i is the capacity of the link i. When the packet pair arrives at the destination, the dispersion is given by the maximum dispersion (Δ_R) between the two probe packets among all the hops crossed by the pair in the path. Hence, Δ_R is expressed by

$$\Delta_R = \max_{i=1,\ldots,H}\left(\frac{L}{C_i}\right) \tag{2.5}$$

Note that the probe packets are stamped upon reception. According to this, the end-to-end capacity can be measured using

$$C = \frac{L}{\Delta_R}. \tag{2.6}$$

This methodology is deployed in bprobe [2], SProbe [18], pathrate and Nettimer [19]. The measurement in bprobe assumes that the dispersion felt in the forward channel is the same as in the reverse channel. In other words, this method allows the source to calculate the end-to-end capacity since it is assumed that the packet pair dispersion does not change when packets come from the sink to the source as well as it assumes that paths and links are both symmetric. On the other hand, SProbe is also a non-cooperative tool which is capable of estimating the end-to-end capacity on both directions (downstream and upstream) by measuring the time dispersion between a pair of generated packets. Finally, the Nettimer is a cooperative tool which performs passive measurements through the analysis of traffic traces.

The quality of the measurements may be affected by the presence of cross traffic. The packet pair technique assumes the absence of cross traffic between probe packets, which may not be true because narrow links tend to delay packets, leading to the decrease of the end-to-end capacity as well as the overestimation of this metric.

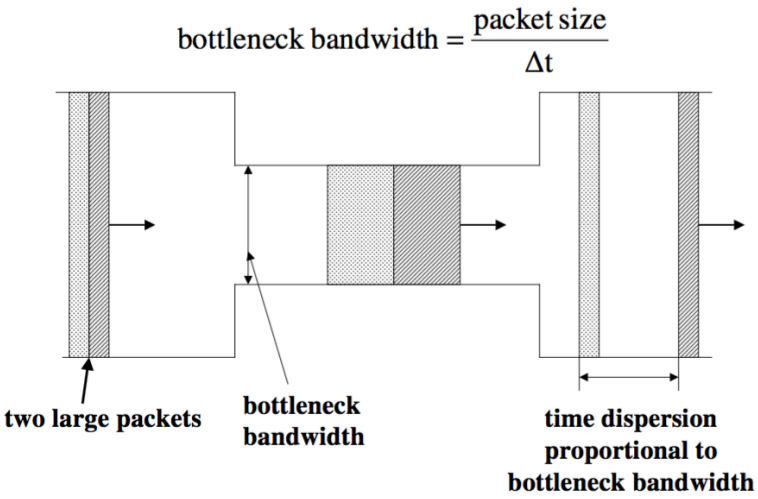

Figure 2.6: Packet pair dispersion [18]

2.2.1.7 Per-hop capacity

This section presents two approaches to measure the capacity of each hop along the path: the one-packet model and the multi-packet model.

One-packet model/Variable packet size (VPS)

The one-packet model [20], also known as Variable Packet Size (VPS) model, consists on sending probe packets to the destination in order to measure the RTT from the source to each hop that composes the path. Through this method it is possible to express the packet delay as a function of the packet size since serialization delay is proportional to the size of the sent packet. The serialization delay, often called transmission delay, is the time required to push all the data to be transmitted from the network card to the transmission medium. It can be expressed by the ratio between the size of the packet and the transmission rate of the link. According to this, the capacity of a link can be determined by calculating the packet size change ratio needed to change the serialization delay. Hence, as illustrated in figure 2.7, the delay to reach node l can be measured by calculating the time interval $t_l - t_0$, which is given by the sum of the transmission delays plus the propagation delays of all the links along the path. The expression is given by

$$T_l = t_l - t_0 = \sum_{i=0}^{l-1} \left(\frac{s}{b_i} + d_i \right), \tag{2.7}$$

where $\frac{s}{b_i}$ represents the serialization delay and d_i represents the propagation delay. Finally, the capacity of each link is denoted by

$$b_{l-1} = \frac{1}{k_l - k_{l-1}}, \tag{2.8}$$

where

$$k_l = \sum_{i=0}^{l-1} \left(\frac{1}{b_i} \right). \tag{2.9}$$

The k_l's values are determined using linear regressions together with the measurement delays of different packets sizes from the source (node 0) to the node l.

Pathchar [20], Bing [21] and clink [22] are examples of non-cooperative tools that implement the VPS technique.

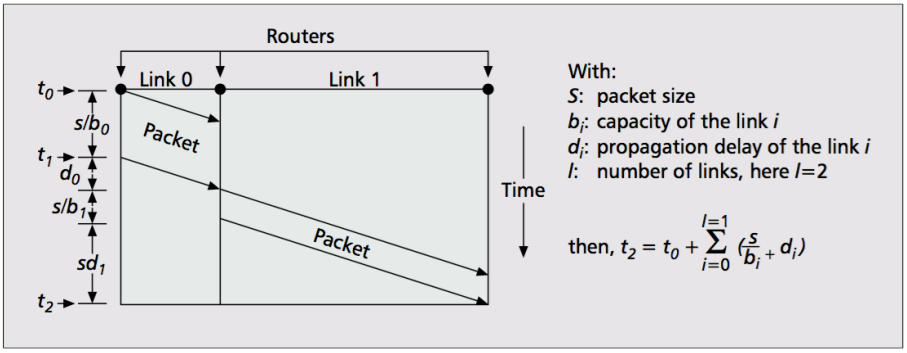

Figure 2.7: Per-hop capacity using one-packet model [5]

Multi-packet model

The multi-packet model [23], or tailgating model, is focused on intra-flow queueing delays. The technique is deployed in Nettimer and consists on sending sequences of two packets, being the first sent with a particular TTL. These two packets have different sizes: the packet $k-1$ is the largest non-fragmented packet and k is the smallest possible one. The methodology assumes that the larger packet (tailgated) has a larger transmission delay. The packets are identified as $k-1$ and k. The $k-1$ packet will be dropped at a specific hop according to the assigned TTL value. On the other hand, the smaller packet (tailgater) will continue without queueing because the first packet has been dropped.

The Nettimer has developed a new feature to cope with ICMP limitations. It uses TCP FIN/RST mechanism to force the receiver to sent back a RST segment in response to a sent tailgater FIN segment.

2.3 Platforms, Tools, Testbeds and Services

Several algorithms were presented in the previous section to provide an overview of deployed methodologies that perform Internet measurement. The present section aims to provide a taxonomy of the existing and emerging platforms [24], tools [25], testbeds [25] and services [25], complementing the information provided so far. Note that in the last few years mobile network measurement has been a focus of research, since nowadays an important method of Internet access is the cellular networks. Hence, a comparison

covering the end-to-end mobile network performance measurements, diagnosis and application prototyping is presented in tools and testbeds as well as in services. Finally, after giving a comprehensive overview of the current efforts, some future work will be presented with the objective of showing the research directions of mobile network metrology.

2.3.1 Platforms

Internet measurement platforms have emerged in the past few years, which have been an important forthcoming to both, Internet Service Providers (ISPs) and users, since they provide an evaluation of the practiced Quality of Service (QoS). They also let consumers confirm whether the Service-Level Agreement (SLA) is being met. Despite the fact that a regular consumer can diagnose their home network using ISP benchmark, it also enables the possibility of the ISP to identify, isolate and fix problems related to the access network. Moreover, the measurements information is also important to network regulators whereas they can compare broadband providers offerings and allow the broadband industry to implement new policies.

The platforms provide operational support by enabling network operators (ISPs) to diagnose and troubleshoot network infrastructure issues. Despite that, Internet access is provided by fixed-line and mobile access where several other measurements are made. Some relevant platforms are examined in this section. Netradar [26] is an example of a mobile access platform which started in 2012 and is operated by Aalto University. It uses a client-server based architecture where clients perform the measurements at the closest server. Servers are implemented in a cloud and globally distributed. Additionally, Netradar performs both active and passive measurements. Active measurements include metrics such as latency and TCP goodput, measured performing upload and download speed tests. Other informations like handovers, signal strength and locations are also measured during the process. Moreover, the research based on this platform studies the correlation between signal strength and network parameters. In [26] Sebastian Sonntag *et al.* have shown that the bandwidth is clearly affected by radio technology and signal strength, being reduced by a third due to the congestion at the base station or even more due to poor provisioning.

Another example of a mobile access platform is Portolan [27, 28], which started in 2012, supporting both active and passive measurements. Latency, forwarding path at the IP and at AS level and bandwidth are examples of active measurements. Otherwise, it passively gathers available wireless networks, signal strength and cell coverage. The mentioned achievable bandwidth is measured by SmartProbe [29], being the forwarding path captured by MDA-traceroute [30], where MDA is the acronym for Multipath Detection Algorithm. Additionally, probing is implemented using multiple sockets in parallel allowing multi-threaded processes. The Portolan architecture is centralized acting as a controller and a server simultaneously. Despite that, it uses regional proxies to perform

microtasks, which are a set of instructions resulting from a Extensible Markup Language (XML) file submitted by the central server. This central server can also push the microtasks into the mobile devices.

Operational support seeks the resolution of end-to-end performance problems. In order to solve multi-domain network issues, a collaborative initiative developed by Energy Sciences Network (ESnet), GÉANT, Internet2, and Brazil's National Education and *Rede Nacional de Pesquisa (RNP)* has deployed the Performance Serive Oriented Network Monitoring Architecture (PerfSONAR). It aims the identification and isolation of problems that affect network paths that sustain scientific data exchange. PerfSONAR [31] measures various metrics such as network utilization, available bandwidth, end-to-end latency, packet loss, connection stability and forwarding path.

Other Platforms such as SamKnows [32], BISmark [33], Dasu [34], RIPE Atlas [35] and RIPE TTM [8] also accomplish Internet measurement by gathering other metrics related with Domain Name Space (DNS), Voice over IP (VoIP), File Transfer Protocol (FTP), Path Maximum Transmission Unit (PMTU), Global Positioning System (GPS), sattelite conditions, etc.

2.3.2 Testbeds

Nowadays Internet access is predominantly made by cellphones. Network testbeds are an usefull resource in order to evaluate the performance of mobile applications. Since real-time user interactions like multiplayer games, video chat or augmented reality have been widely deployed in the past few years, a low latency network service is desired to deliver user requests between mobile devices and cloud datacenters. There are previous studies which have shown that a significant variation in end-to-end mobile network performance affects the application request delay.

Additionally, end-to-end systems that emulate communication protocols to create performance profiles based on factors like signal strength, geographic location or network provider, need to be aware of user privacy policies, because they might be abusing of contributed user resources. It is possible to classify these approaches as uncurated and curated network testbeds. Uncurated testbeds perform users experiments without the necessity of Institutional Review Board (IRB) approval process. An example of an uncurated testbed is Mobile Internet Testbed for Application Traffic Experimentation (MITATE) which has started in April of 2013 at Montana State University (MSU). MITATE [36] is innovative since it allows application prototyping traffic experiments between mobile hosts and back end server infrastructure. It performs active measurements and proposes solutions to security and mobile resource sharing problems by separating traffic generation from application code execution and through tit-for-tat mechanisms, which protect the mentioned resource sharing. After downloading the mobile application, the user needs to register his credentials to create a MITATE account. Hence, the functionality works as illustrated in figure 2.8. Firstly, it starts with step 1 where a user creates an

experiment by uploading a XML configuration file to the database which can be queried for experiments in step 2. In step 3 the data defined by the experiment is transferred from the device to the measurement server when a criteria like geographic location or network type is met. In step 4, traffic metrics and network metrics together with metadata are reported to the database. Finally, in the last step, the user has the option of accessing Web interface in order to visualize, or download the experiment data collected by multiple devices.

On the other half, curated testbeds with an IRB approval to their experiments may collect privacy information such as user's traffic or location history. PhoneLab [37] was started in 2013 at University of Buffalo North and is an example of a programable network testbed. It is unintrusive and it focuses three main aspects, namely overall energy breakdown, opportunistic charging and 3G to Wifi transactions, being the latter illustrated in figure 2.9. Moreover, it runs Google Android Opensource Smartphone Platform (AOSP) with the testbed integrated into the Operating System (OS), enabling the device to collect operational data (e.g phone status and battery level) as well as custom application log data, being both uploaded to PhoneLab servers during device charging. However, this option leverage no scalability since it depends on the data plan subsidy provided to the users as well as it depends on the phone's hardware, which may become slow to support the testbed. There are other University projects that deployed network testbeds, not presented in this section [25].

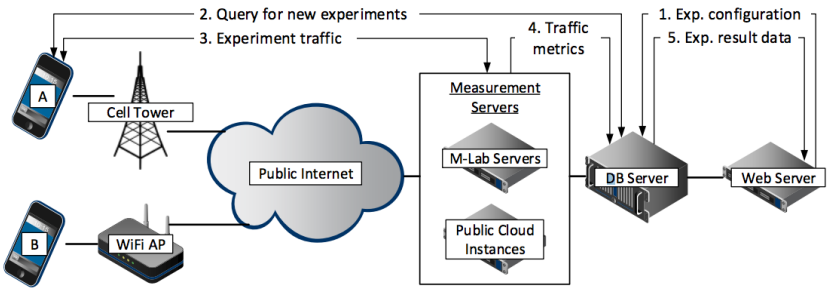

Figure 2.8: MITATE architecture and steps of a network traffic experiment [36]

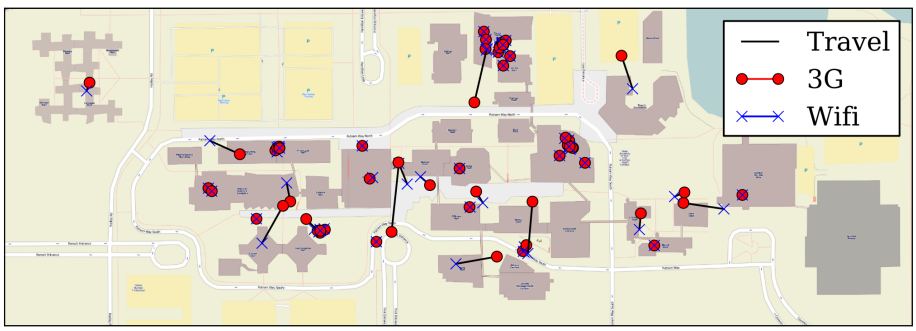

Figure 2.9: Campus map of University of Buffalo North indicating where the 3G to Wifi hand-offs occur [37]

2.3.3 Tools

There are several methods and metrics to measure network performance. Measurement tools have been developed by industry, research and regulators. These tools may vary in how they gather network metrics and in how they select the measurement devices. Consequently, the major difference between a testbed and a tool is that a testbed is limited to a certain set of experiments. Otherwise, a tool is able to perform wide-scale network monitoring during a long period, which may be useful to the mentioned use fields. A brief introduction to some tools is made in this section, providing an overview of the existing utilities. It covers iPerf, ping, OWAMP, traceroute, SProbe and Pathload.

IPerf [38] is a cross-platform command line tool which performs active measurements. It can gather multiple metrics related with timing, buffers and protocols in both IPv4 and IPv6. For each test it reports the bandwidth, loss, and other parameters depending on the protocol that is being used. For example, in TCP it is able to report Maximum Segment Size(MSS)/MTU size and observed read sizes, being jitter an example of UDP metric.

Ping is a command line tool that measures RTT by calculating the difference between the time an ICMP echo packet is sent from the source and the time the ICMP reply packet arrives at the same source.

OWAMP is an active measurement tool which measures one way delay by inserting UDP test streams with negotiated features (packet size, number of packets, timestamps) into an IP network. It is formed by two different protocols, OWAMP-Control and OWAMP-test, where OWAMP-test takes cares of transmitting probe packets between two measurement points. On the other hand, the OWAMP-Control is responsible for measurement sessions and for fetching their results. Note that OWAMP is a standard tool that gathers IPPM metrics in an interoperable way.

Traceroute, which is a tool previously described, compiles the list of hops travelled by an UDP packet including how long each hop takes from the source to the destination. It completes the list by taking advantage of TTL field which is defined to stop at a specific hop in order to generate an ICMP message, which contains the hop address. Thus, this technique is fully performed when an ICMP response from the sink reaches the source.

SProbe is a tool that takes advantage of the TCP behavior. In TCP, a SYN packet is answered with a RST packet when the SYN packet is sent to an unactive port. Hence, it is possible to determine the time dispersion of a channel in both directions by computing the difference in time at which the data packets were received. These data packets were sent in response to the HTTP GET request transmitted to the Web server. Although this approach is able to work in uncooperative environments, it has two relevant drawbacks: first, the firewalls block SYN packets targeting inactive ports; and second, the tool is slow and unscalable. According to this, a new approach inspired in PBProbe (namely SmartProbe) have emerged. SmartProbe is a tool developed for smartphones with an energy-saving perspective, which estimates bottleneck link capacity between two hosts.

As illustrated in figure 2.10, after the initial handshake between the Estimator (E) and the Prober (P), the k value, representing the minimum length of packet train that will be sent by the Prober in each RTS message, is calculated. The estimator computes the relative delay sum S_i and dispersion D_i as soon as it receives correctly a packet train. Otherwise, the train is invalidated at the timeout expiration. The network is assumed as congested whether three failures are experienced, resulting in this case in an experiment restart with half of the train length.

Pathload is a non-intrusive tool that uses UDP periodic streams for probing the network together with a TCP control channel for exchanging messages with the stream features between the two end points. Pathload available bandwidth measurements are comparable with the ones presented by MRTG test.

There are other utilities, such as Pathchirp, IGI and Spruce, which were mentioned in the algorithms section. Besides, there are other tools like Nettimer that implement Internet measurement but are not described in this state of the art.

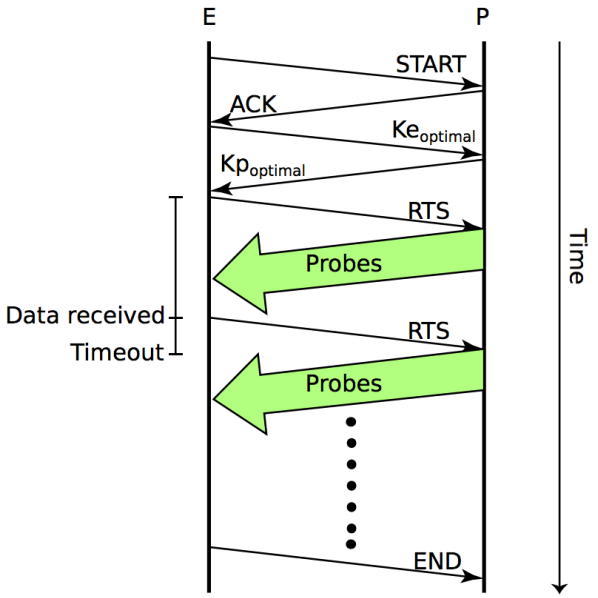

Figure 2.10: SmartProbe protocol [29]

2.3.4 Services

There are also services that perform network monitoring as well as network discovery and diagnosis. This section introduces three services: OpenSignal, Vodafone NetPerform and Netalyzr. They offer insight to developers, researchers, regulators and network operators. Indeed, many of the deployed services are closed-source and the data collected by them is not publicly available.

OpenSignal [39] is a mobile application released in 2013 that supports active and passive measurement. Passively, it runs periodically measurements sending posteriorly the data to the OpenSignal servers. Actively, it also records multiple metrics. In order to estimate the download throughput, eight concurrent HTTP GET of 108 Mb files for each request are sent to a CloudFront's CDN replica. The tests run during a certain period, after which the application is able to calculate the average download speed. The same principle is applied in the upstream direction, where multiple HTTP POST requests are sent to the Amazon AWS server with the objective of uploading small images. Equally, the HTTP HEAD method allows latency measurement by sending those requests to www.google.com. Furthermore, OpenSignal is able to gather network-related information to the Wifi as well as data associated with cellular networks, such as cell tower ID, location area code or even system ID if the device is connected to a cellular network. The major drawback of this deployment is that it is not capable of detecting the presence of traffic prioritzation or shaping. Moreover, hundreds of megabytes are needed to perform throughput measurements, which can be complicated to users with low data plans.

Vodafone NetPerform [40], which is a paid service in many countries, was started in 2014 with the purpose of understanding the amount of data used by the applications installed on customer's smartphones. Thus, it allows Vodafone to troubleshoot connectivity and bandwidth issues demanded by those applications. When it comes to functionality, at every hour a TCP connection is established with the Vodafone server in order to measure metrics such as latency variation or uplink/downlink rate. These metrics together with information collected from the cellular network, which includes signal strength, device location or the quality of 2G/3G/4G coverage, enable Vodafone to diagnose and resolve device related network issues. For example, by allocating high capacity bandwidth for services that have that kind of necessity. Similarly to OpenSignal, Vodafone NetPerform is not able to verify whether ISP is performing traffic shapping.

Finally, Netalyzr [41] service is a diagnostic tool that characterizes several issues related with connectivity, performance and security. Netalyzr is able to record diverse data, including Network Address Translation (NAT) detection, port renumbering, packet fragmentation, path MTU, Wi-Fi/cellular configuration, network topology and other metrics related with DNS and Transport Layer Security (TLS). Netalyzr detects the presence of NAT by comparing public and private IP adresses, identifying how addresses and ports are renumbered. It is also able to detect HTTP and DNS proxies by sending requests to the back-end servers. In the former, the proxy is detected by examining the HTTP responses sent by the server for signs of modifications. The latter applies the same principle, since response parameters such as transaction ID or public IP address are verified. In both cases, a modification found means that a proxy exists.

To conclude, the future effort remains on the development of new tools that meet the requirements among developers, researchers, network operators and regulators. Simultaneously, an enhancement of the existing utilities should be made in order to develop new approaches and capabilities.

2.4 Summary

The state of the art related with *Internet measurement* is summarized in the following table, which contains the described algorithms and concepts with their relevant features as well as the tools and metrics related with those algorithms and concepts.

Algorithms/Concepts	Features	Metrics	Tools
One-way delay[7]	•Implies synchronization in both end-systems •Calibration determines instruments errors	One-way delay variation (jitter), One-way delay	OWAM[8], iPerf[38], ping
Packet loss[9]	•Loss distance can gather the spacing between loss periods •Measurement in forward and reverse channels	Loss distance, Loss period, noticeable loss rate	OWAMP
Packet reordering[10]	•Source infer if both packets and ACKs were reordered in flight through single and dual connection tests •SYN test - uses three way handshake to verify the reordering	One-way reordering in both directions	OWAMP, QoSMet
Route	•UDP packets to trace the route	Ordered IP addresses hops	traceroute[42]
Packet train dispersion[2]	•In the absence of cross traffic ADR is equal to end-to-end capacity •ADR metric[14] is useful to monitor the QoS of the path	Asymptotic Dispersion Range (ADR), end-to-end capacity, end-to-end available bandwidth	cprobe[2], pathrate[14]
Self-loading periodic streams (SLoPS)[15]	•Monitor one-way delay variations •Silent period between streams maintains probing traffic less than 10% of the available bandwidth	End-to-end available bandwidth	Pathload[16]
Probe rate model[5]	•Search for turning points to detect available bandwidth •Chirp train only needs $2n-2$ packets using a packet-pair technique to exploit $n-1$ packet spacings	End-to-end available bandwidth	Pathload, Pathchirp[17]
Probe gap model[12]	•If the sending gap between packets (Δ_{in}) is smaller or equal to the transmission time it is operating under Joint queueing region (JQR) condition •Δ_{in} is optimal when the average output gap is equal to the initial gap	End-to-end available bandwidth	IGI[12], Spruce[13]
Multi Router Traffic Grapher (MRTG) test[13]	•Implements Simple Network Management Protocol (SNMP) •Reduces signalling and router load by predicting autonomously the available bandwidth through Available Bandwidth Estimator (ABEst)	End-to-end available bandwidth	MTRG[13]
Packet pair dispersion[2]	•Maximum dispersion (Δ_R) between two probe packets among all the hops crossed along path gives the dispersion of that pair	End-to-end capacity	bprobe[2], SProbe[18], pathrate, Nettimer[19]
Variable packet size (VPS)/One-packet model[20]	•It is possible to express the packet delay as a function of the packet size since serialization delay is proportional to the size of the sent packet	Per-hop capacity	Pathchar[20], Bing[21], clink[22]
Multi-packet model[23]	•The tailgated packet ($k-1$) is dropped at a specific hop due to its TTL while the tailgater (k) will continue without queueing since the tailgated have been dropped	Per-hop capacity	Nettimer

Chapter 3

Client-Server Architecture

3.1 Introduction

The client-server model has been widely used to support services in the context of distributed systems, since it is possible to build a network architecture in which a client is physically separated from a server. The client is an instance that determines when the session starts and ends, by sending requests to a server which is awaiting for new requests in order to process them and send out responses.

The architecture proposed in this dissertation follows the client-server design. The server side was developed in Java, being the client side deployed both in Android and in Java, to be able to perform measurements in any TCP/IP network including cellular networks, such as 3G and 4G. The project is publicly available in the GitHub site, in the Uniform Resource Locator (URL) `https://github.com/glazeni/`. According to this, three BTC estimation algorithms were deployed in order to perform the measurements tests, which are executed during 32 seconds on both directions. Note that this measurement time is related to the regulations imposed by United Kingdom (UK) regulator Ofcom, on which the 3G/4G performance is evaluated by fixed time transferences of 30 seconds (transferring part of a 2 GB file) to test the downlink, and of 15 seconds (transferring part of a 100 MB file) to test the upload speed [43]. Hence, 2 extra seconds were used on each test with the objective of having at least 32 bandwidth measurement values, allowing to validate if it is possible to design an approach compatible with Ofcom's. Moreover, as a support to the results validation, after the deployed BTC algorithms run, the command line iPerf3 tool [44] is launched directly from the code, with the purpose of having the least possible variation of the network conditions.

Hence, as a consequence of the developed work, a protocol was implemented in order to coordinate the tests, which defines the communication flow in this architecture,

illustrated in figure 3.1. Three different algorithms were implemented. The first was the packet train dispersion [2] adapted to the TCP scenario. The second and third were two different approaches of bulk transfer capacity estimation, named as Sample Second Thread and the Sampling Read Time. These algorithms are described in detail in the fourth chapter.

This chapter explains the client and server applications implementation. It also presents the configuration setup used to perform the measurements.

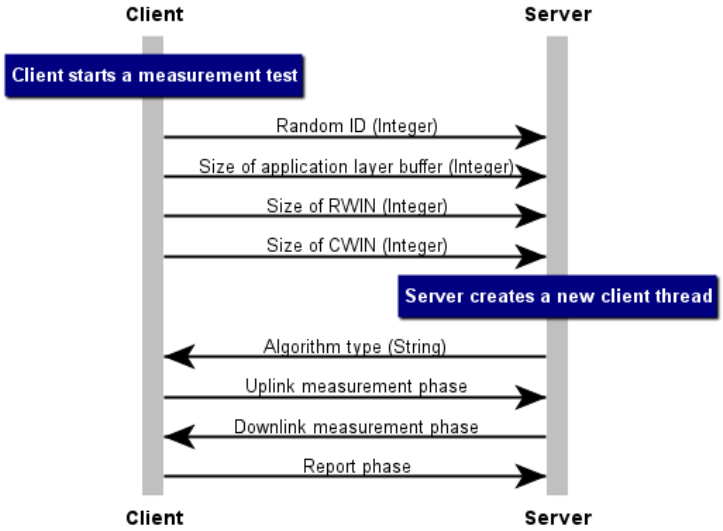

Figure 3.1: Implemented measurement protocol - sequence diagram

3.2 Server Side

3.2.1 Operation overview

The server has a ServerSocket binded to port number 20000, listening for connection requests. At the moment that the connection is accepted, the server reads the client Identification (ID) in order to verify whether the client is registered or not. If it is a new client, a random integer ID is assigned and the server receives the measurement parameters defined by the client, such as the size of buffer upon send and reception, the TCP congestion and receiver windows which will configure socket buffers, along with a binary option, that specify if the Nagle's algorithm is turned on or off. Hence, the client socket, denominated as the socket binded to a new client thread, is configured in the TCP classe's properties, using the parameters received by the client plus the blocking read operation timeout option, and the setSoLinger option which guarantees TCP operation to be completed when set to false, since the connection will be closed only when the data transmitted to the socket has been successfully delivered. Before the client thread initialization, the multithread server checks if the maximum of thirty connections has

been achieved, launching a new thread if it is not. Attending to the server specifications, which will be presented on section 3.2.3, a maximum of 30 simultaneous users was defined in order to avoid the system overload.

When the client thread is initiated, the server sends the measurement algorithm type to the client in order to perform the test. The general procedure is the same in the three presented algorithms: the three use three different TCP connections to perform the upload, the download as well as the report phase on which the client sends the results to the server, which means that in practice, three different sockets are used by each end point.

After the test is completed, the client sends a report to the server containing the results from the downlink measurement along with the bandwidth results obtained by iPerf tool.

3.2.2 Java deployment

In this section the Unified Modelling Language (UML) class diagram corresponding to the server architecture, which is illustrated by figure 3.3, is presented and explained in order to provide an entire comprehension of the developed work. Note that, the UML class diagrams presented along this chapter were generated under the UML Lab software [45]. Additionally, the Graphical User Interface (GUI) is illustrated by the figure 3.2.

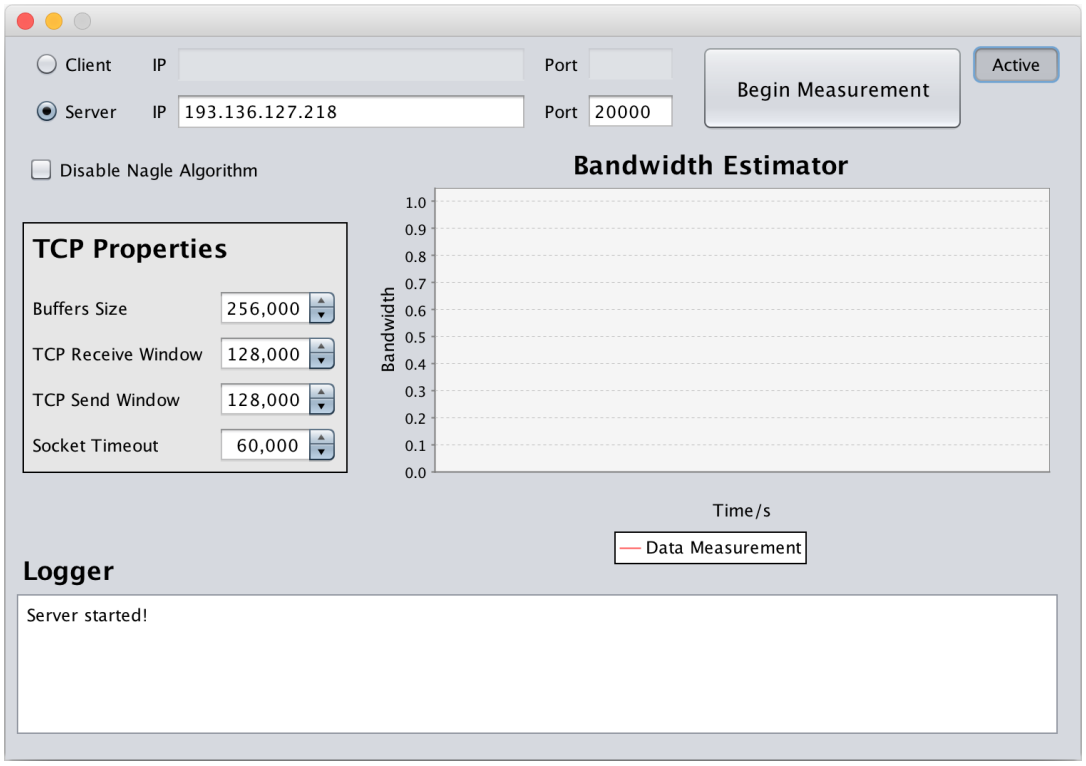

Figure 3.2: Java user interface

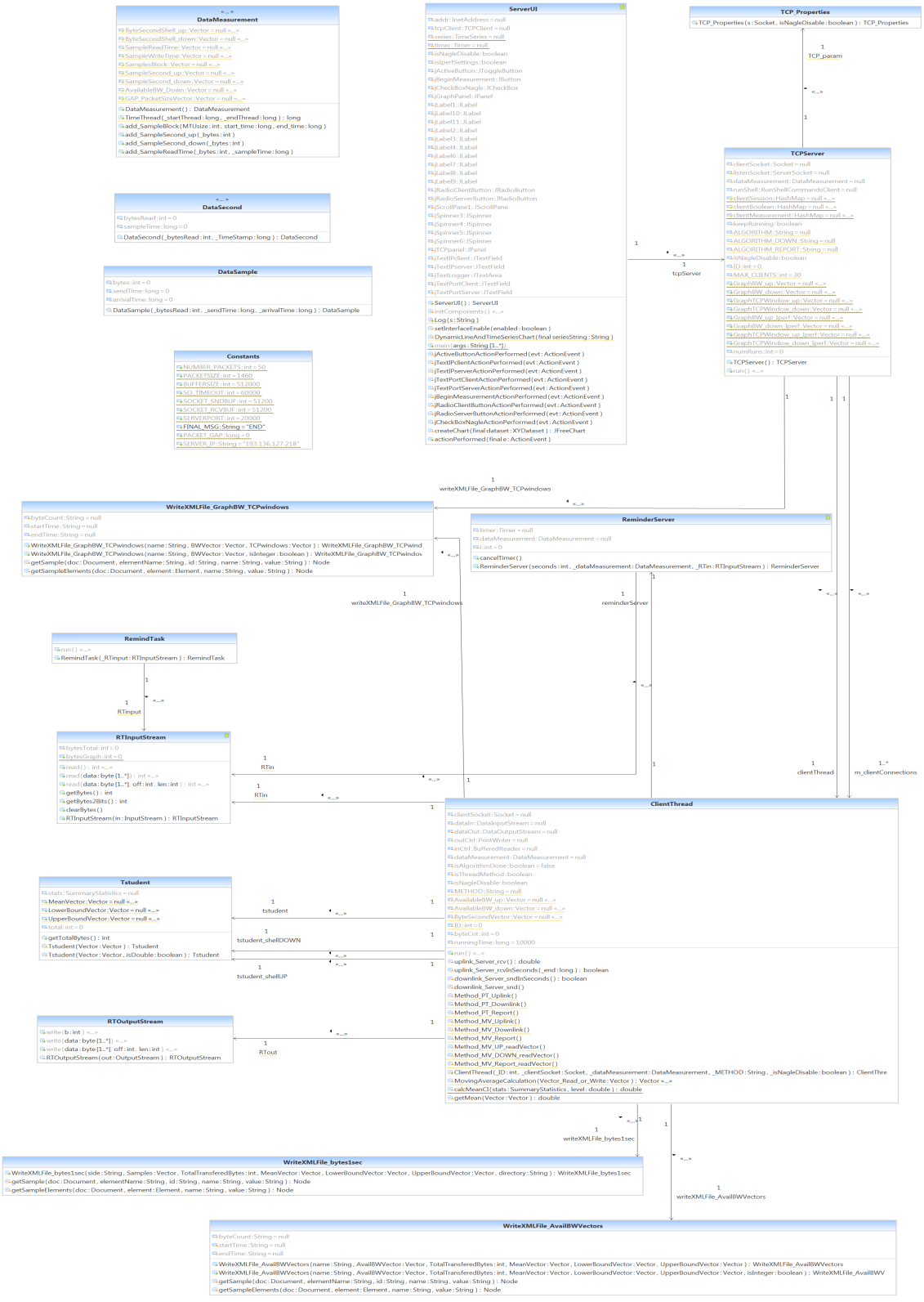

Figure 3.3: TCP Server UML class diagram

The main method of this project is located on the *ServerUI* class, which represents the server User Interface (UI), where the new *TCPServer* instances are created through the click of the active button on the corresponding interface, as illustrated in figure 3.2. The *TCPServer* class has a socket configured with the parameters specified by *TCP_Properties* listening on port 20000 to let clients connect. When the *ClientThread* is initiated, the server sends the algorithm type to the client, which will start the test. At this point, one of the three available methods will be selected, starting the uplink measurement part. Only the uplink measurement is started because it uses a different TCP connection than the downlink, being still used a third connection in order to report the results back to the server.

The *RTInputStream* and the *RTOutputStream* are respectively the classes responsible for reading and writing bytes on the socket, being the reading class a subclass of *FilterInputStream*, while the writing class is a subclass of *PrintStream*. It is important to notice that several tests were performed in order to choose the Java I/O classes. It was found that the *FilterOutputStream* is a bottleneck that does not allow reaching throughput values above 20 Mbps as is illustrated in figure 3.4, obtained on the downlink direction in a Gigabit Passive Optical Network (GPON) network. The throughput measured using the two algorithms was much lower than the one measured using Iperf.

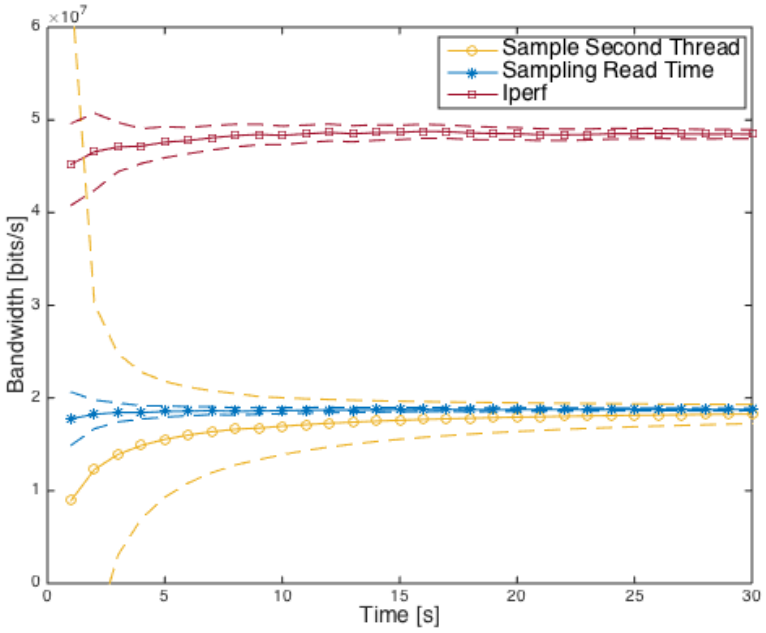

Figure 3.4: Downlink bottleneck of *FilterOutputStream* class in GPON scenario

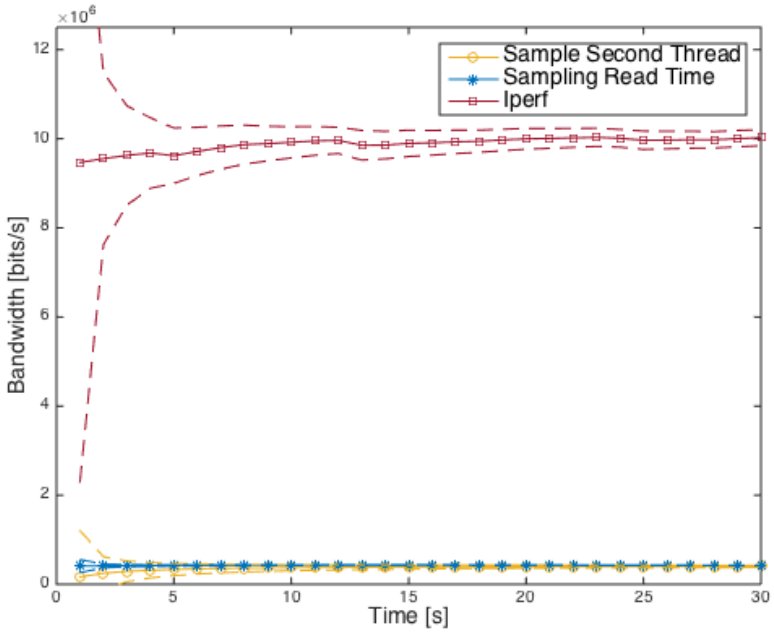

Figure 3.5: Uplink bottleneck of *FilterOutputStream* class in 4G scenario

Moreover, in the Android environment, due to the inherent hardware limitations of the mobile phone, the throughput restraint was even more noticeable, with a measured bandwidth of 500 Kbps on the uplink direction, while with the *PrintStream* class, the Android application was able to measure up to 11 Mbps, as is shown in chapter 4.

The *DataMeasurement* class is where the samples are maintained. Between algorithm phases, new instances of *ClientThread* are created, which require a separate class to keep measurement vectors independently for each client. After the uplink measurement is completed, the socket binded to this phase of the test is closed. At this moment, the client starts a new TCP connection to the server. When the server receives the incoming connection, it verifies if the client is already registered on the server, and starts a second TCP connection to perform the downlink test. When downlink measurement is accomplished, a third socket is binded to the same client ID. In this report phase, besides the downlink results, the client also sends the iPerf results obtained for both directions. When the server has all the results, the Student's t-distribution confidence interval for mean values with unknown standard deviation is calculated using the Apache Commons Math API, [46], being then exported to a XML file, which contains the measurement test results along with the mean, lower bound and upper bound values calculated by *t-student* class. Note that, each measurement direction will generate two independent .xml files corresponding to the results acquired by the Java application and by the iPerf tool. The exporting procedure is handled by the *WriteXMLFile_bytes1sec* for both bulk transfer capacity estimation algorithms, namely Sample Second Thread and Sampling Read Time.

Now that the general operation is described, the architecture differentiation is related

with the selected algorithm. Methods *Method_MV_Uplink*, *Method_MV_Downlink* as well as *Method_MV_Report* are part of the implementation of the Sample Second Thread algorithm, which sends buffers of random bytes during 32 seconds in order to test the maximum capacity of TCP channel. After the measurement period, the server interrupts the connection, closing the socket. During the connection, the transferred bytes are measured by a parallel thread (*ReminderServer*) which starts a timer that expires on every second, calling the *RemindTask* class which reads the number of bytes transmitted during that second and saves it into a vector defined in *DataMeasurement*.

The other bulk transfer capacity estimation method, called Sampling Read Time, is defined in *Method_MV _readVectorUP*, *Method_MV_readVectorDOWN* and *Method_MV_Report_readVector* classes, which do not use a parallel thread. Instead, it records the time and bytes received on every chunk. After the test completion, the *MovingAverageCalculation* function iterates over that vector in order to find how many bytes have been transferred in each second.

The third method, denominated as *Method_PT*, implements the packet train dispersion algorithm. This is a method originally built on top of UDP, which was adapted to TCP in MobiBand project [47]. Thereby, this implementation uses PrinterWritter [48] to send a train of 50 packets constructed with pseudo random characters. Upon reception, the BufferedReader [49] is used to read each line printed by the sending side. Note that in this method, both end points record the time at which the burst starts and ends.

Finally, the .xml files are exported by the classes with *WriteXMLFile* denomination to be posteriorly analysed in MATLAB [50] software, through the *xml2struct* [51] script, which converts .xml files into a MATLAB structures in order to access the data. This analysis will be explained in detail in the fourth chapter of this dissertation.

3.2.3 Measurement tests setup

The server uses a public IP address (193.136.127.218), which corresponds to the domain of tele2.dee.fct.unl.pt, listening on port 20000. Note that the client TCP connections are binded to the port 11008, being the traffic posteriorly redirected to the port 20000 of the server, which is an host in the referred network. Simultaneously, an iPerf3 server is running, listening on port 20001, which receives the traffic originated from the port 11010 of the NAT.

It should be also noted that both servers, the Java application server developed in the context of this dissertation, and the iPerf server are running in the same physical machine.

3.2.3.1 System specifications

- Operating System: Xubuntu 16.04 LTS
- Memory RAM: 8 GB @ 1333 MHz
- Processor: Intel(R) Core(TM) i5-2320 CPU @ 3.00GHz
- Network Card: RTL8111/8168/8411 PCI Express Gigabit
- Graphics Card: Intel Corporation 2nd Generation Core Processor Family Integrated Graphics Controller
- Storage: 320GB WDC WD3200AACS-0

3.3 Client side

In this section, the client side of the architecture is presented. As mentioned in the introduction of this chapter, the client can be a mobile phone through the Android deployment, or a computer implemented under the Java environment. Note that in both cases the same Java module, corresponding to the class diagram demonstrated in figure 3.6, was used.

3.3.1 Java deployment and operation

This section presents the Unified Modelling Language (UML) class diagram corresponding to the client architecture, illustrated by 3.6, developed under the Java environment.

Figure 3.6: TCP Client UML class diagram

In this implementation, similarly to the server side Java project, the main method runs on *ServerUI_Client*, which is similar to the server interface, but adapted to the client side. At the moment that the user clicks on active button, the interface will be available, being the measurement test started when the begin measurement button is pressed. Hence, a new instance of *TCPClient* is initiated, starting a *Connection* with a socket configured according to the parameters present in the *TCP_Properties* class, which is also part of the server architecture.

After the socket creation, the client sends the parameters to the server informing about send and reception TCP window size, the buffer size as well as the state of the Nagle's algorithm.

Next to this initial parameters exchange, the client receives the method from the server. Analogously to the description made in the Java deployment section of the server side, the packet train dispersion algorithm has a behaviour different from the algorithms

used to estimate the BTC. Consequently, and thinking inversely, the client performs the exact same operations of the server, but in the reverse order. If the client is sending at the maximum capacity of the TCP channel, the server is receiving, measuring the uplink capacity; if the server is sending at the same maximum capacity, the client is receiving allowing the download measurement. For this purpose, the same *RTInputStream*, *RTOutputStream* and *DataMeasurement* classes are used both at the client and the server.

The noticed difference in the two BTC algorithms is related to the way the methods record the bytes transmitted on each second interval. As referred for the server side, the Sample Second Thread uses a parallel thread with a timer that expires every second, while the Sampling Read Time algorithm obtains the bytes sent by iterating over a vector that contains all the received chunks. These chunks allow knowing when a second has passed, since the number of received bytes are recorded with a timestamp.

Moreover, the deployed algorithms begin a new connection from inside the *Connection* class to perform downlink measurement after the uplink test is concluded. The report phase is started when the downlink part of each algorithm is concluded, in order to report the downstream results to the server. The server is awaiting for incoming connections to instantiate a new thread, which can be a new thread for a known client ID to perform downlink or report phases, or can be a completely new client with a new identification number.

Finally, the *RunShellCommands* class is used to run commands on terminal directly from Java code, allowing the accomplishment of iPerf measurements tests.

3.3.2 Android deployment and operation

3.3.2.1 Android overview

Android is an open-source operating system initially developed by Open Handset Alliance (OHA), which is consortium of hardware, software and telecommunication firms that share a goal of advance open standards for mobile devices [52].

As presented in figure 3.7, Android platform stack has at the bottom the Linux kernel, which is responsible for basic system functionality. For instance, process management, memory management and power management. On top of the Linux kernel, is located the Hardware Abstraction Layer (HAL), which bridges the hardware and the software, allowing android to be agnostic about lower-level driver implementations. The libraries layer is above HAL and contains the native operating system libraries written in C and C++, as well as the Android Runtime (ART), which was introduced as an experimental feature in Android 4.4, providing the compilation of application byte code into machine code upon the installation. Over this layer and before the application layer, where the user interacts, is situated the application framework that manages the basic functions of the device, such as view system, resource management, voice call management, activity management, notification management and content providers [53].

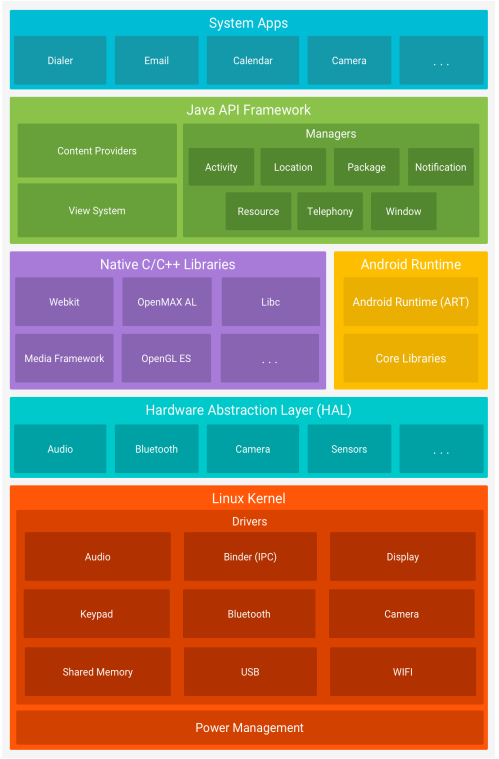

Figure 3.7: Android operating system architecture [53]

3.3.2.2 Model-View-Controller

Android applications are designed according to Model-View-Controller (MVC) pattern, on which all objects are classified as model objects, view objects or controller objects.

A model object is responsible for holding and managing data related with application, which are generally custom objects created by the developer such as .xml files, JavaScript Object Notation (JSON) files or SQL data. A view object handles the input-output interaction with the client, like touches on the screen, being the result of conjugation between .xml files from the model and the Java classes from the controller. Thus, the controller objects tie the view and model objects, representing the logical functionality of the system. Typically, in Android a controller is a subclass of an activity, a fragment or a service, which are defined as fundamental components, where the content providers, the views and the intents are also included [54]. The figure 3.8 presented below illustrates this explanation, giving a visual perception of the addressed subjects.

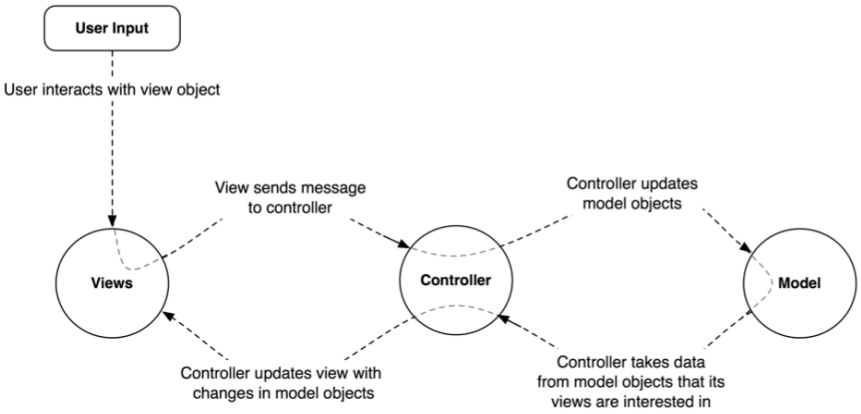

Figure 3.8: Model-View-Controller [54]

3.3.2.3 Internet measurement App

In this section, the architecture of the Android application developed under the scope of this dissertation will be explained in detail using figure 3.9, which is a diagram of the developed project.

As demonstrates figure 3.8, the controller is the bridge between the model and the views, which are the result of the Java controller operation over the .xml model files. In this project, the controller is located in Main and in Fragment packages, being the model handled by the Resources directory. Hence, the referred two packages contain the Java classes responsible for updating the model, which is defined by the .xml files present in the layout folder under the Resources directory.

Starting with the Main package, it contains the *MainActivity.java* which is the only activity subclass in the project, since the rest of the development was made with subclasses of fragments. The *MainActivity.java* uses a *TabLayout*, which provides a horizontal layout to display tabs, as denotes one of the application figures from 3.11 to 3.14 presented below. This *TabLayout* is linked to the *CustomViewPager.java* in order to reflect adapter changes, scroll state changes, and clicks from one in the other. However, before the linking, the *CustomViewPager* needs to supply the views for this pager, which is achieved by *PagerAdapter.java*, where the fragments are associated with each tab in the *TabLayout*.

The Fragment package is divided into four other packages, corresponding each package to his respective tab. As a result, the *FirstFragment.java* is the class where the measurements tab, presented on figure 3.11, is defined. Consequently, in this tab the user performs the measurement tests by pressing the *Begin* button, which will start a new instance of *TCPClient.java*. The TCP Client package is the same Java source code described in the beginning of the client side section, with the exception of the subclass of *TCPClient.java*, since instead of a thread an *AsyncTask* is used. Asynctask class permits the execution of background operations and the publishment of the results on the UI thread without having to manipulate threads or handlers. In order to display the evolution of

the running test, a ProgressBar is used, indicating what direction is being measured.

The second package is related to the results that are displayed on the screen as instances of *Data.java* through a *RecyclerView*. A *RecyclerView* is basically a more complex list view, which is classified as a subclass of *ViewGroup*, and similarly to the parent class, it contains child view objects. The entity responsible for binding this child view objects to *Data.java* items is implemented in *MyAdapter.java*, being this a subclass of *RecyclerView.Adapter*. Moreover, the *SecondFragment.java* also contains a graph [55] that is updated in real time when the download stream is being tested. In addition, as figure 3.12 demonstrates, the results contain the connection type, the measurement algorithm used by the server, the current date, the average received bytes and the ping result.

On the other hand, as figures 3.13 and 3.14 show, the third tab of the application concerns about network information and TCP Properties, being both nested fragments of *ThirdFragment.java*. Either one is accessible through the options menu at the upper right corner, where the *FragmentManager* handles the transaction of views between the *ThirdFragment.java* and one of his childs. Since in this application the fragments are exclusively used to handle the user interface, a *ChildFragmentManager* is used for placing and managing the nested fragments within the third fragment.

At last, the fourth fragment designated as "About" keeps the information related to the application development.

Additionally, it is relevant to refer *InternetMeasurement.java* which is a subclass of Application, where a static method from *TypefaceUtil.java* is invoked, in order to replace the original Android font to the Eurostile font [56] placed in the Assets directory.

Besides the layout folder, which contains the .xml files that will become view objects due to Java controller classes, the Resources directory also maintains every piece of application that is not code. Accordingly, the drawable folder, contains the background image as well as all the icons presented in the app, except for the Android Application Package (APK) icon shown in figure 3.10, which is located in the Mipmap folder. The Menu folder contains the options menu items of *ThirdFragment.java*, being the Values folder the place where other resource types such as, colours, strings, styles, dimensions or attributes are defined.

Finally, at the moment that the build process starts, the *AndroidManifest.xml* which contains the essential meta-data about the application, will provide that information to the Android system in order to generate the .apk file successfully.

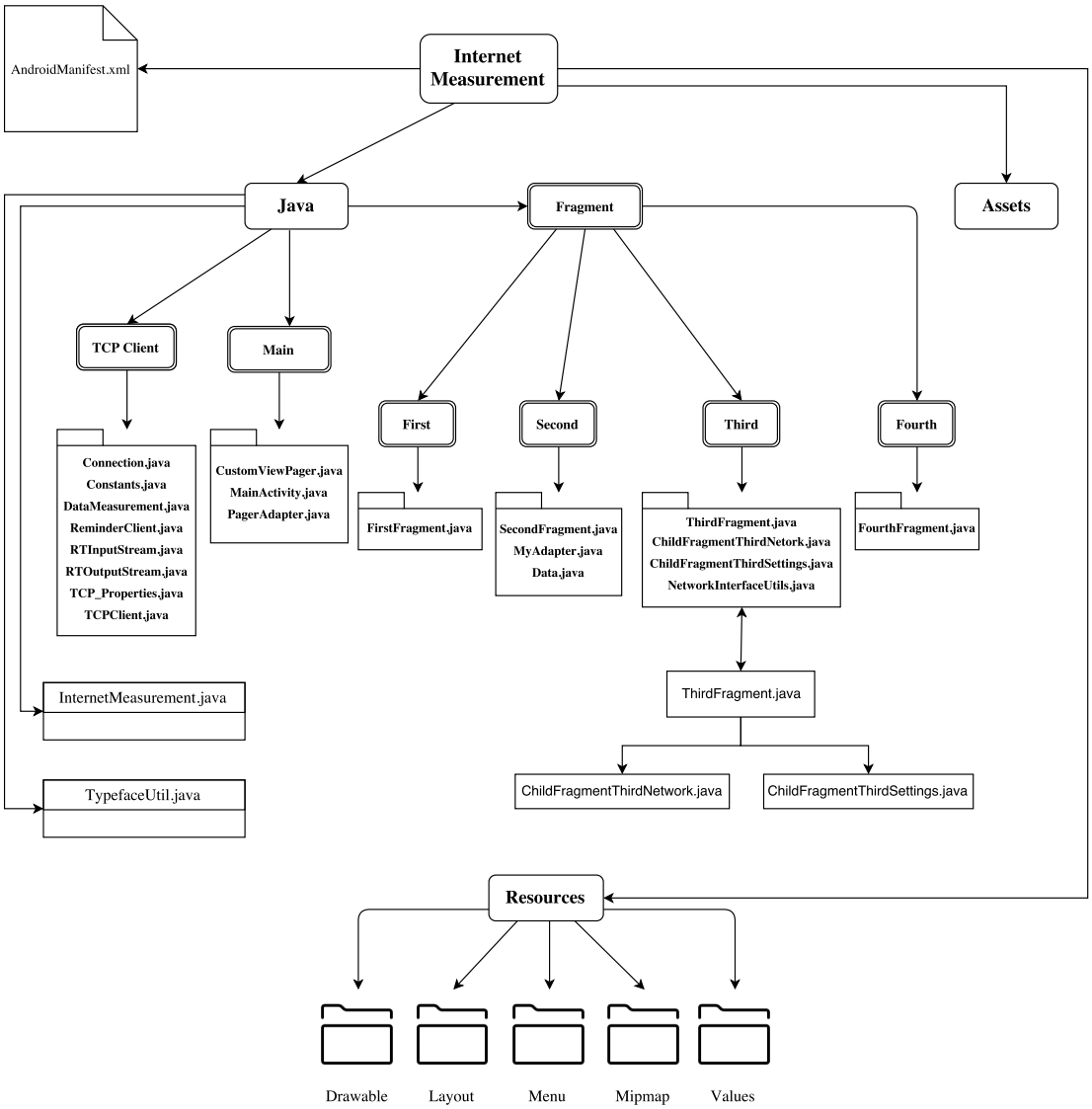

Figure 3.9: Android project diagram

Figure 3.10: Internet measurement app logo

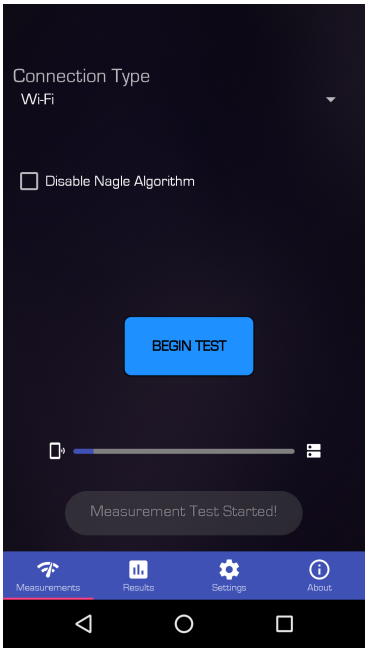

Figure 3.11: Android Application first fragment tab

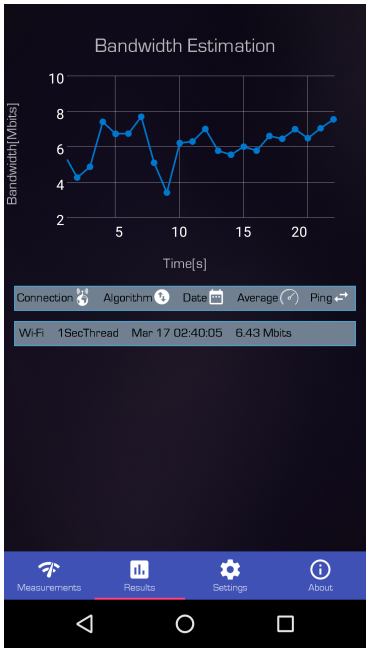

Figure 3.12: Android Application second fragment tab

Figure 3.13: Android Application third fragment tab

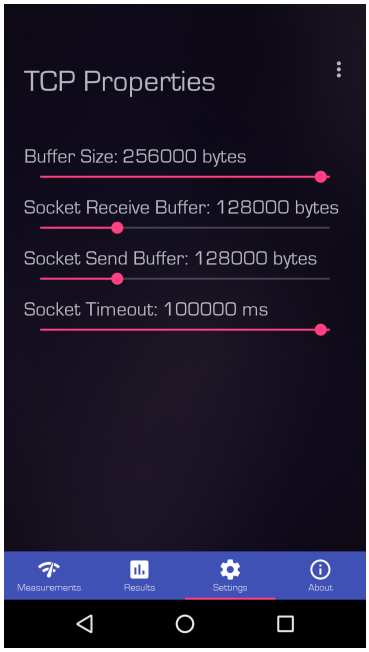

Figure 3.14: Android Application fourth fragment tab

3.3.3 Measurement tests setup

The tests were performed by the deployed application using a TCP connection to the port 11008, while the iPerf measurements use the port 11010, being the traffic redirected by the NAT as explained in section 3.2.3.

The results that will be presented in the next chapter, were obtained using a Nexus 5 mobile phone and an emulated computer system, since it was not possible to run shell commands directly from Java code under the macOS environment. Hence, the system specifications englobing the virtual machine used by the computer, and the mobile phone, will be described below. Moreover, note that the ping result shown by the Android application is not the RTT related to the tele2.dee.fct.unl.pt but to tele1.dee.fct.unl.pt, which is an host belonging to the same subnet.

3.3.3.1 System Specifications

Android

- Operating System: Android 6.0.1 Marshmallow
- Memory RAM: 2 GB @ 1600 MHz
- Processor: 2.26 GHz Quad-Core Processor, Qualcomm(R) Snapdragon(TM) 800 MSM8974 Chipset
- Compatible Networks: 2G GSM (MHz) 850, 900, 1800, 1900 / 3G UMTS (MHz) 850, 900, 1700, 1900, 2100 / 4G LTE (MHz) 800, 850, 900, 1800, 2100, 2600
- Battery: 3.8 V 2300 mAh,
- Storage: 32 GB

Java - Computer

- Operating System: macOS 10.12.3
- Memory RAM: 16 GB @ 1600 MHz
- Processor: Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz
- Network Card: RTL8111/8168/8411 PCI Express Gigabit
- Graphics Card: AMD Radeon R9 M370X 2048 MB /Intel Iris Pro 1536 MB
- Storage: 512GB APPLE SSD SM0512G

Java - Virtual machine

- Operating System: Xubuntu 16.04 LTS
- Memory RAM: 2 GB @ 1600 MHz
- Processor: Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz (2 processor cores)
- Network Card: RTL8111/8168/8411 PCI Express Gigabit
- Storage: 21GB SCSI Disk

Chapter 4

Algorithms and Results

4.1 Introduction

The main objective of this dissertation is the estimation of bulk transfer capacity (BTC), which is defined as the maximum throughput achievable by a single TCP connection [57]. This metric is obtained by calculating the average number of bytes transmitted per time unit. Contrarily to capacity or available bandwidth, which are network observables, BTC is hard to measure because it is defined in terms of TCP throughput, which is affected by many factors, such as path latency, competing traffic, router queueing policy, buffer sizes, random losses, among other [4].

Although the available bandwidth is a metric independent from transport control protocol, BTC depends on how a single TCP connection is affected by concurrent flows [3]. According to this, BTC is directly related with TCP's congestion control algorithm which manages the throughput of each TCP connection. Thus, the RTT and the TCP buffers, denominated as congestion window (CWIN) at the sender and receiving window at the receiver (RWIN), are factors that define throughput.

Three algorithms are presented in this chapter, starting with the descriptive operation of the method, followed by the gathered results. In order to have a valid evaluation, the results obtained in the scope of the developed work, are compared with iPerf tool, which is a reference in bulk transfer capacity estimation [38]. Two .xml files are generated in each direction containing in the file name the client identification number (ID), the type of the algorithm, the direction as well as the date timestamp.

Consequently, the extensible markup language files contain the measurements and the total transferred bytes, along with Student's t-distribution confidence interval for mean values with unknown standard deviation. Moreover, lower and upper bounds relative to the mean values are calculated in increasing intervals from $[0,1]$ to $[0,32]$,

corresponding the last period to the total test duration.

In addition, a study showing the evolution of the bandwidth in function of TCP windows is presented for BTC estimation algorithms. On the other half, a study concerning the influence of the size of the packets and gap between them in the measured throughput was made for packet train dispersion algorithm, showing that the method did not converge to a valid value with the current implementation.

Finally, the graphical representation of the results is obtained in MATLAB environment using the conversion of .xml files into structures through *xml2struct* script [51].

4.2 Algorithms description

As mentioned in the introduction of this chapter, throughput is controlled by the underlying congestion control mechanism. A well-known throughput bound is represented in 4.1 (in bps), related to the buffers used by the TCP protocol and the RTT,

$$Throughput \leq \frac{8 \times min(CWIN, RWIN)}{RTT} \quad [58], \tag{4.1}$$

An application layer buffer of 256 KB is used to perform the read/write operations on both sides. Moreover, in the transport layer, a buffer corresponding to the TCP windows is used for sending and receiving, with a size of 128 KB, which is also the default TCP buffer size used by iPerf tool [44]. Thus, the deployed methods measure the end-to-end BTC by calculating the average number of bytes that have been transmitted per second, between two end-points of a network. The first algorithm, called Sample Second Thread, reads the bytes transmitted during a period of a second by using a parallel thread with a timer that is scheduled to expire when 1000 ms have passed. The second algorithm, named Sampling Read Time, records every chunk of received bytes with the respective timestamp on a vector, which will be posteriorly iterated in order to obtain the number of transferred bytes upon each second.

After the execution of the functions responsible for handling the packet transmission, the report phase is performed, followed by the close of the socket that is assigned to this part of the algorithm, which will finish the method. Thus, in this last phase, the server receives the download results along with the iPerf estimations on both directions, calculating the mean and the 90% confidence interval, which will allow to have a lower and an upper bound for the respective mean value. Note that, as referred in the introduction of this chapter, the mean and the confidence intervals are calculated in increasing intervals from $[0,1]$ to $[0,32]$. Moreover, for a given independent sample with unknown variance, a $100(1-\alpha)\%$ confidence interval for the population mean is defined as:

$$\bar{X} \pm t_{1-\frac{\alpha}{2},N-1} \frac{s}{\sqrt{N}} \tag{4.2}$$

where $\bar{X}$ is the sample mean, s is the sample standard deviation, N is the sample size, α is the desired confidence level, and $t_{1-\frac{\alpha}{2},N-1}$ is the $100(1-\alpha)$ percentile of Student's t distribution with $N-1$ degrees of freedom [59]. Note that (4.2) is an approximation only valid for independent samples, which according to the law of large numbers tends to a normal distribution when $N \to \infty$.

It is possible to validate the measurements, comparing the confidence intervals obtained by the developed application and the confidence interval of the iPerf tool.

4.2.1 Method to reduce BTC measurement overhead

In the context of active measurement tools development, the amount of traffic injected into the network is a concerning issue. Hence, in this dissertation a method to reduce the BTC measurement overhead is proposed regarding a stop criteria of 5 stable slots. It consists on computing the Jain's fairness index [1] for all the subsets with n consecutive records on both directions, and normalized such that they range from 0 to 1. The measurement slots were considered stable at the first subset with a Jain's fairness index higher than 0.999, which indicates homogeneity and low dispersion among the samples.

4.3 Study of packet train dispersion algorithm as a function of packet size and sending time gap

As described in the chapter two of this dissertation, the packet train dispersion[2] is a technique to obtain the available bandwidth by sending N ICMP echo request packets of size L, where $N > 2$. The dispersion, denoted by $\Delta(N)$, is the amount of time between the reception of the first packet and the last packet. Consequently, the available bandwidth is obtained by dividing the number of bytes sent by the elapsed time between the first and the last bits received as expressed by equation 2.3. However, in the scope of this dissertation the packets are transmitted using TCP, as adopted by the open-source MobiBand project [47]. A study evolving the packet size and the sending time gap between the packets in the train is presented in this section. The present study was developed in an ADSL Internet access link.

The figures 4.1 and 4.2 demonstrate the evolution of the bandwidth measured as a function of the packet size. The study was started with a value of 512 bytes for the uplink and 1460 bytes for the downlink, which correspond to the largest sizes of the datagram packets captured by the Wireshark network analyzer software [60]. Several tests were performed, with no time gap, varying the size of the packets to ten times the initial value, being the size multiplied by a factor of two after each measurement. Figure 4.2 shows that, it is possible to perform a correct measurement test as long as a suitable size is

selected for packet. However, on the uplink direction, it could not be obtained, given that, values of packet sizes below the MTU would be required.

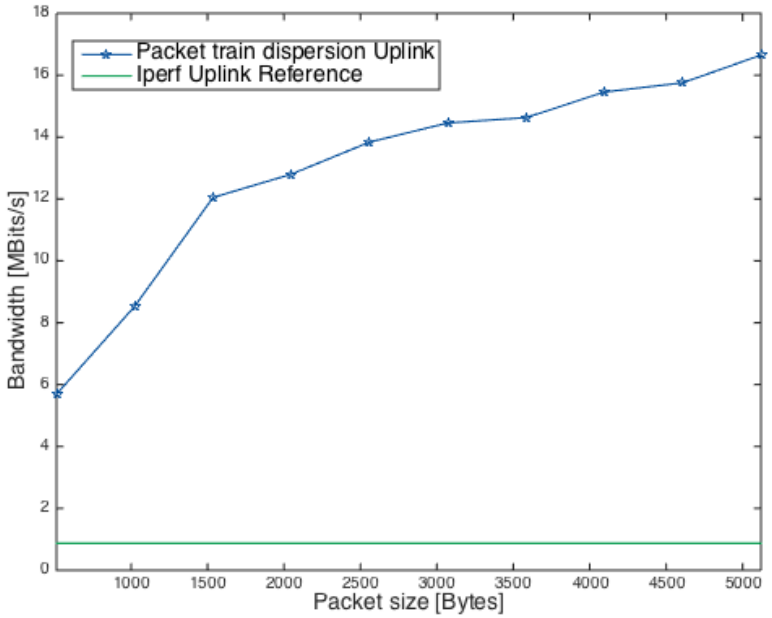

Figure 4.1: Study of packet train dispersion as a function of packet size on the upload

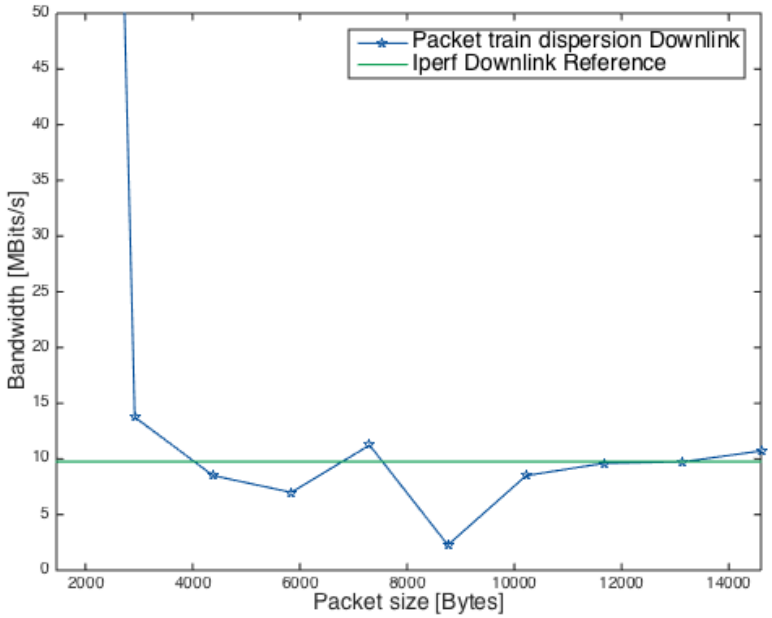

Figure 4.2: Study of packet train dispersion as a function of packet size on the download

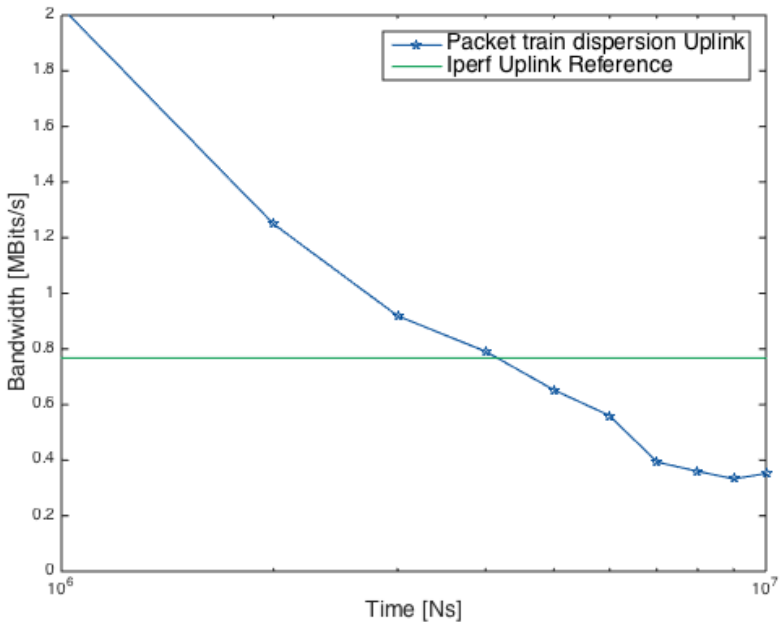

Figure 4.3: Study of packet train dispersion as a function of sending time gap on the upload

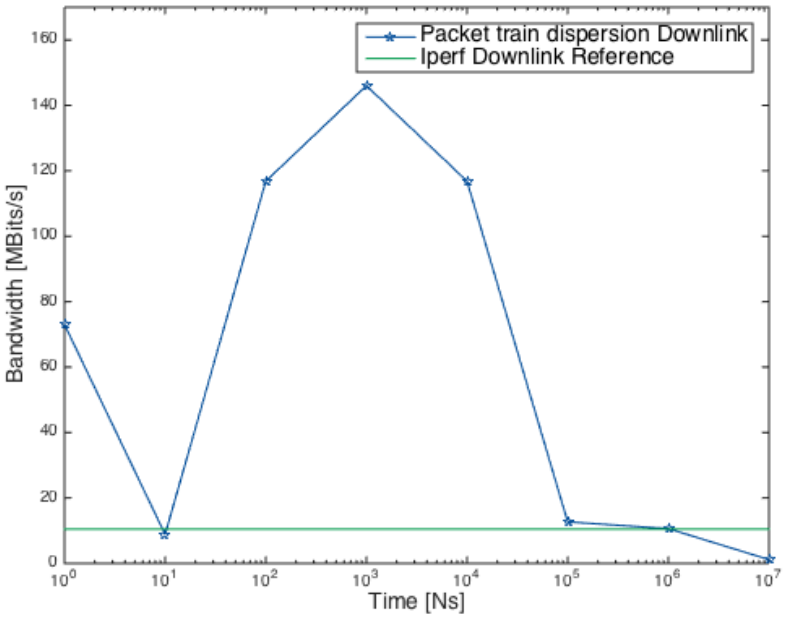

Figure 4.4: Study of packet train dispersion as a function of sending time gap on the download

Since this algorithm consists on sending a train of probe packets, the time gap between probes on the train is a determinant factor that is studied in figures 4.3 and 4.4. The gap between the packets was varied, in order to find a suitable time that allows to measure the available bandwidth correctly. Note that in the upload, the variation was made from 0 to 10 ms while in the opposite direction a range of $[10^0, 10^8]$ nanoseconds was used. Consequently, on the uplink direction the reference bandwidth would be measured using a 4 ms time gap. On the other half, the download demonstrates that with higher throughput values it is possible to find more suitable time gap values. On other tests not reported in this thesis, for different network conditions, different suitable packet sizes and gap values would be required. So, no optimal configuration was found that fits different network environments. Attending to the high influence of these parameters on the available bandwidth estimation, it was decided not to use this method, since it does not show a convergence to the BTC value measured by the iPerf3 tool.

4.4 Study of BTC algorithms bandwidth as a function of TCP windows size

In order to analyse what would be the optimal TCP window size or buffer for the two BTC measurement methods considered, a study concerning the evolution of the bandwidth as a function of TCP windows size was conducted, showing the trade-off between the size of TCP buffers and the measurable bandwidth. Note that this study was performed for both the uplink and downlink directions. The results illustrated by the figures 4.5 and 4.6, were obtained over a GPON optical fiber network using a fixed application level read/write buffer of 512KB.

The study was developed by varying the TCP buffer size from 1 KB to 512 KB in successive multiplications of the last window value by a factor of two. As expected, the larger the TCP buffers size, the higher the bandwidth. Note that in order not to limit the throughput, the value used for the read/write buffer corresponds to the last value of the TCP window size, which assures that the TCP window will never be greater than the read/write buffer.

Therefore, these results represent the maximum throughput measured on both directions, using the measurement test setup described in sections 3.2.3 and 3.3.3, which were respectively 21.98 Mbps and 17 Mbps for the uplink and downlink using the Sample Second Thread method, 22.95 Mbps and 19 Mbps for the uplink and downlink using the Sampling Read Time method, for a TCP buffer of 512 KB. Notice that the maximum throughput also depends of the RTT and network congestion, and that higher values of RTT lead to lower throughputs.

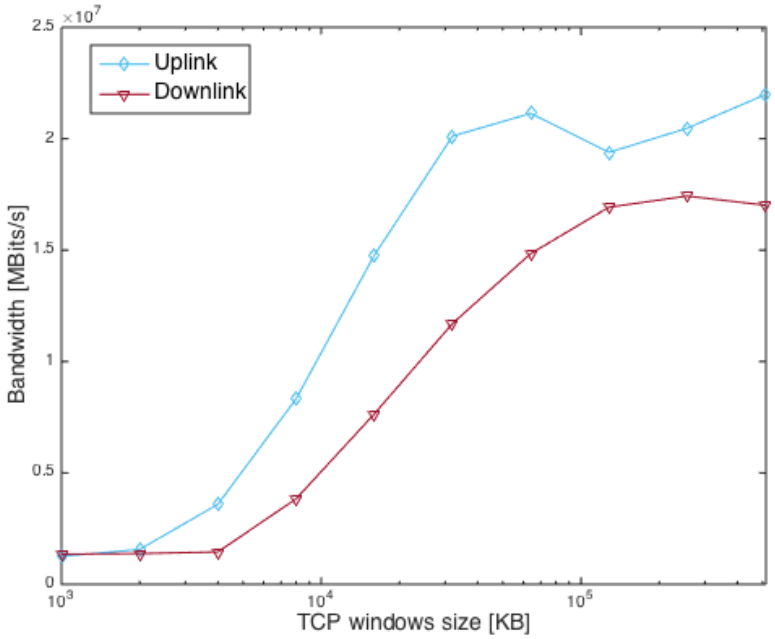

Figure 4.5: Study of Sample Second Thread algorithm

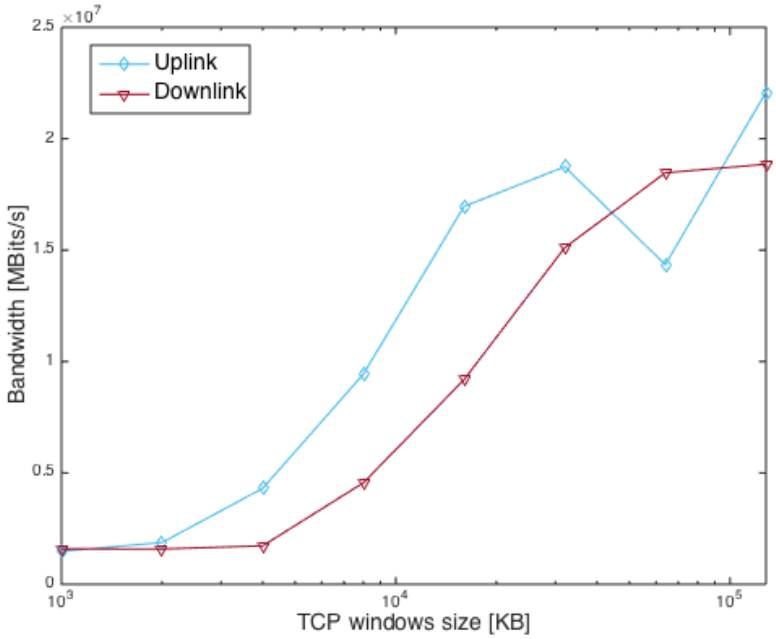

Figure 4.6: Study of Sampling Read Time algorithm

4.5 Results

This section presents the evaluation tests of the bandwidth as a function of time, performed for the uplink and downlink directions. These measurements were made for a Wi-Fi scenario in Asymmetric Digital Subscriber Line (ADSL) and GPON networks, and for 3G and 4G cellular networks. Note that the mobile measurements were performed for the major three Internet Service Providers (ISP) operating in Portugal, being a comparative benchmark between these also presented.

Additionally, in order to distinguish the network conditions and find possible bottlenecks, two environment informations are provided for each test: the RTT obtained by the nping tool [61] through the generation of TCP packets to the port 11008 of tele2.dee.fct.unl.pt domain; and the path crossed by the packets, gathered by the traceroute tool.

Finally, the transferred bytes of each scenario are also shown, allowing to perceive the quantity of traffic that has been injected on the network for each algorithm.

4.5.1 Wi-Fi - ADSL network

The results shown in this section were gathered in a network with an average RTT at the time of the measurements equal to 31.959 ms. Note that the client access was through Wi-Fi, to a router connected to a ADSL modem. Moreover, the path with the corresponding transit delays gathered by the traceroute tool was the following:

1. dsldevice (192.168.1.254) 36.410 ms 46.363 ms 18.159 ms
2. 2.96.54.77.rev.vodafone.pt (77.54.96.2) 188.570 ms 64.937 ms 237.973 ms
3. 113.41.30.213.rev.vodafone.pt (213.30.41.113) 77.225 ms 73.287 ms 76.664 ms
4. fccn.as1930.gigapix.pt (193.136.251.1) 85.051 ms 68.637 ms 76.532 ms
5. router60.10ge.cr2.lisboa.fccn.pt (193.137.0.28) 79.859 ms 100.316 ms 84.829 ms
6. 193.137.124.161 (193.137.124.161) 57.706 ms 147.347 ms 204.222 ms

The next graph plots illustrate the average bandwidth measured as a function of time, corresponding each interval between points in the x-axis to a second. In this network, the upstream bandwidth values obtained for both methods are around 1 Mbps, after the algorithm stabilization, as shown in figures 4.7 and 4.8. The stabilization period of the deployed algorithms depends among other factors of the competing flows and the, slow starting period, which is the initial phase of the TCP congestion control algorithm. In this phase, the congestion window (CWIN) growth is exponential since it is incremented per received ACK. Note that the congestion window limits the sender's amount of data that can transit the network before receiving an acknowledgment. This is also limited by the receiving window (RWIN) maintained at the reception side. Therefore, the lowest of

these windows is the major factor that to controls the throughput of a TCP connection, as denoted in equation 4.1.

The slow start phase is most of the times followed by the congestion avoidance period where the increment of the CWIN is additive, starting this part of the algorithm at the time of a packet lost. Notice that, in figure 4.9, until the seventh second is observed a growth on the *Java Nagle ON* graph, which could be caused by the increase of the congestion window, being in that case the seventh second the moment that a packet is lost.

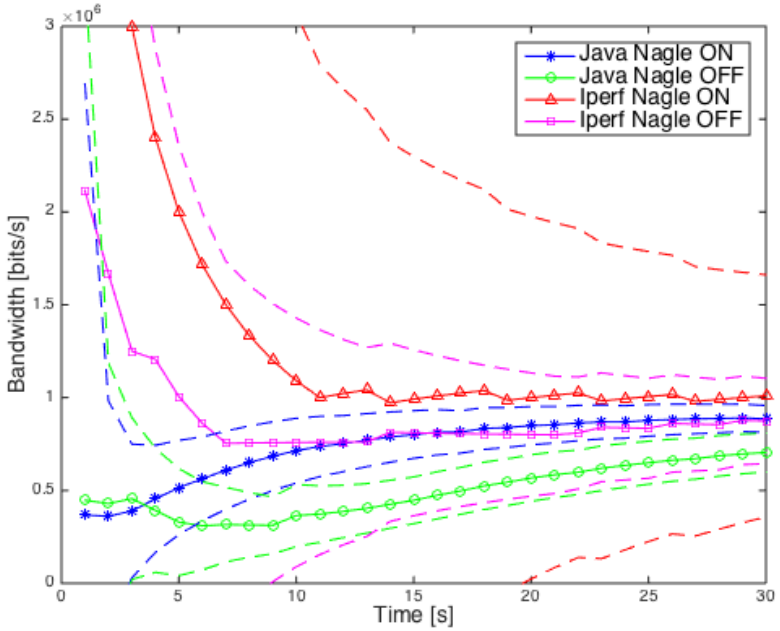

Figure 4.7: Sample Second Thread algorithm upload in ADSL network

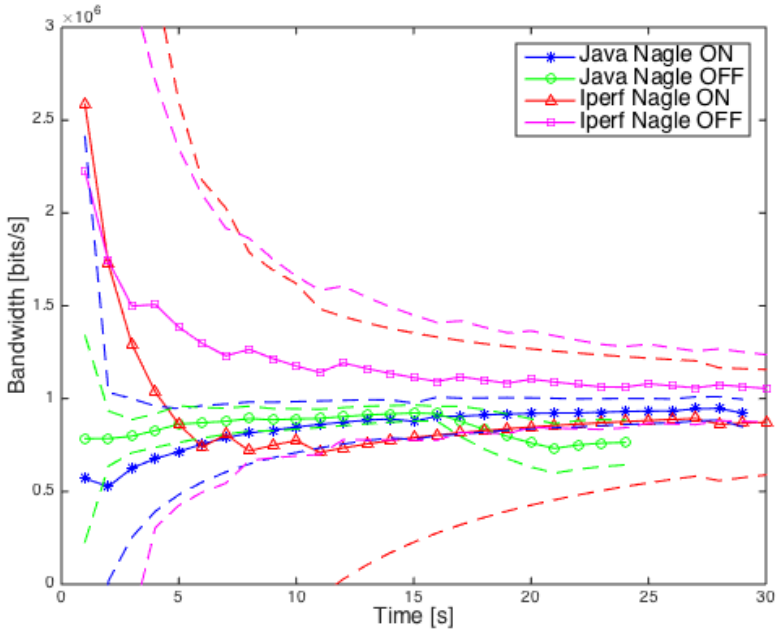

Figure 4.8: Sampling Read Time algorithm upload in ADSL network

The upload results are more stable, since the transmission rate is lower. In the downlink the bandwidth has a higher variance before stabilizing, and this variation is more visible when the Nagle's algorithm is disabled. Nagle's algorithm [11] is a process to increase the efficiency of TCP protocol by favoring the multiplexing of multiple short written or read blocks per datagram packet, thus reducing the number of bytes sent over a network.

Therefore, as expected, higher bandwidth values are obtained when the Nagle is turned on, which is enabled by default in the majority of the systems.

Finally, it is relevant to denote that, in the Sample Second Thread algorithm (figures 4.7 and 4.8), the mean throughput measured using the Java application with the Nagle's algorithm turned on is lower than the iPerf tool, which is exactly the opposite for the Sampling Read Time algorithm (figures 4.9 and 4.10). In the case of Sample Second Thread, the explanation could be a consequence of Java accuracy clock, which depends to the underlying operating system (OS), being the measured time unit in the tens of milliseconds depending on the OS [62]. On the other half, for the Sampling Read Time algorithm, a possible cause for this phenomenon might be the variation of the network load at the time of the measurements with iPerf tool. Moreover, in this method it can be observed that when the Nagle is disabled, the algorithm does not have bandwidth values for the whole 30 second period. This is a consequence of the algorithm implementation, which iterates over a vector that contains chunks of read bytes together with the timestamp. Possibly, a measurement overhead related with the packets timestamp, may be limiting the throughput, since in the Sample Second Thread algorithm the *Sytem.currentTimeMillis()* method

is not executed upon reception, as occurs in Sampling Read Time.

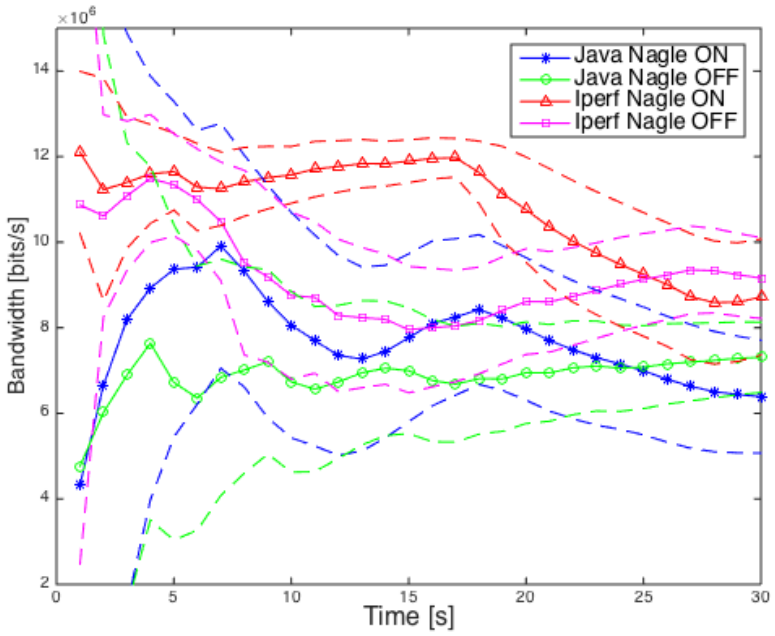

Figure 4.9: Sample Second Thread algorithm download in ADSL network

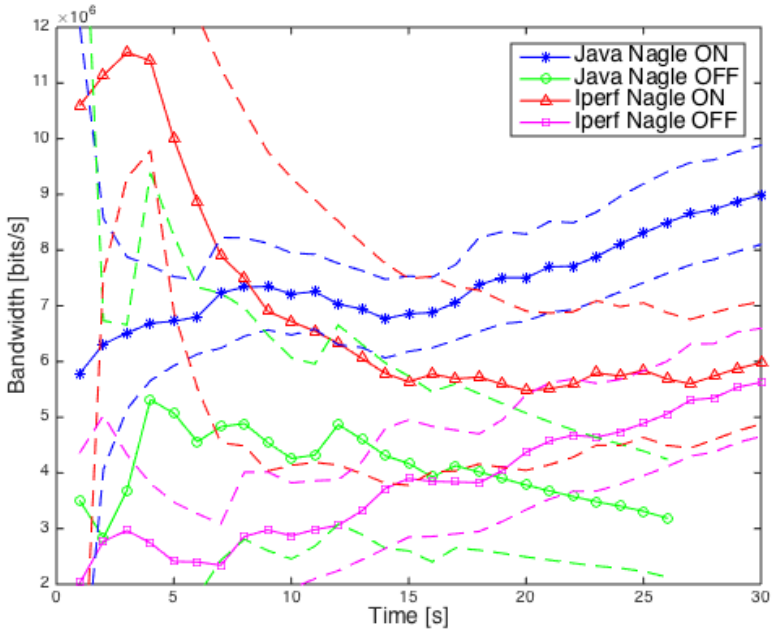

Figure 4.10: Sampling Read Time algorithm download in ADSL network

By analysing the results it is possible to verify that on the uplink direction, the error bound of the deployed application is significantly lower than the presented by the iPerf3 tool, which is not observed on the opposite direction, where the error bound is similar to

the one demonstrated by iPerf. In terms of measured values, as illustrated by figures 4.9 and 4.10, after the stabilization of the congestion control algorithm, the measured value has an error close to the 2 Mbps in the best case, which can be explained by the Java clock inaccuracy clock which has a resolution of the tens of milliseconds.

4.5.2 Wi-Fi comparison for ADSL network

This section presents an evaluation of the deployed BTC algorithm considering a Wi-Fi scenario connected to the Internet using ADSL. The evaluation is performed for both directions, being the comparison with Iperf3 results made with the Nagle's algorithm enabled. The amount of bytes sent by each tool is displayed.

As demonstrated by the figures 4.11 and 4.12, the results obtained by the developed tool are valid since there exists an intersection between the error bounds of both tools. Figure 4.11 shows that the deployed application has an error bound narrower than the iPerf3 tool. However, there exists a divergence between the two BTC algorithms and the iPerf tool, which can be partially explained by the variation of the load in the network, since the measurements were not performed at the same time, being the iPerf estimation made after the BTC algorithms. Note that the tests were not executed simultaneously because the main goal of this work is to measure the maximum capacity of a TCP channel. If two TCP connections were used, the BTC measurement would be limited by the additional concurrent flow.

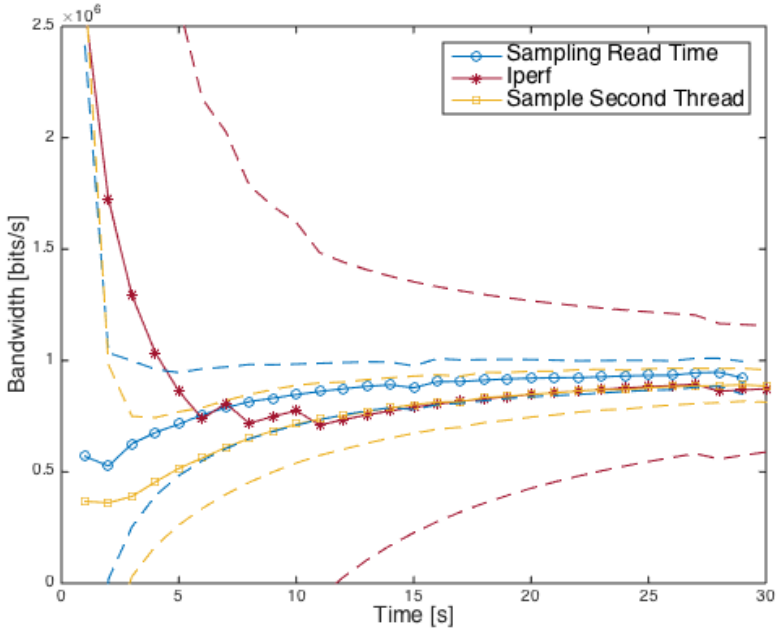

Figure 4.11: Algorithm comparison uplink

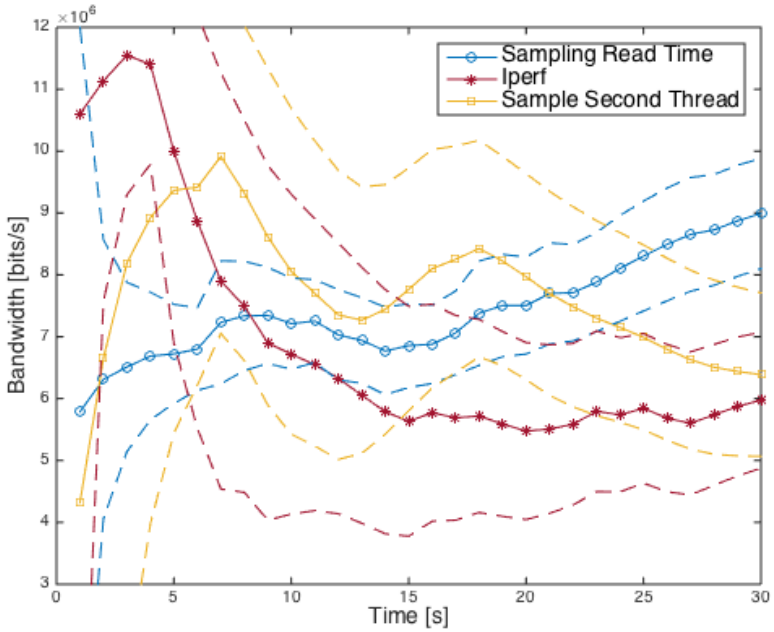

Figure 4.12: Algorithm comparison downlink

An evaluation between the two BTC algorithms concludes that the Sample Second Thread seems to be better, since is the one that presents results more similar with the iPerf tool. Moreover, it can be seen that its final bandwidth values have a small difference to iPerf in all the illustrated measurements.

In matter of network injected traffic, the figures 4.13 and 4.14 placed below, illustrate the number of transferred bits by the two deployed BTC algorithms when compared to the iPerf tool. For this purpose, the Jain's fairness index stop criteria was applied for $n = 5$ stable slots.

In terms of accuracy, the measured error between the deployed algorithms and the iPerf, was calculated for the referred stable slots using:

$$\%\epsilon = \left| \frac{\text{iPerf value} - \text{BTC algorithm value}}{\text{iPerf value}} \right| \times 100, \tag{4.3}$$

Figure 4.11 presents an average upload accuracy of the considered 5 stable records of 78.1419% for the Sample Second Thread and 80.9684% for the Sampling Read Time upon comparison with the iPerf tool. On the downlink direction, figure 4.12 shows an accuracy of 65.6999% for Sample Second Thread algorithm, while solely 40.8521% were obtained for the Sampling Read Time.

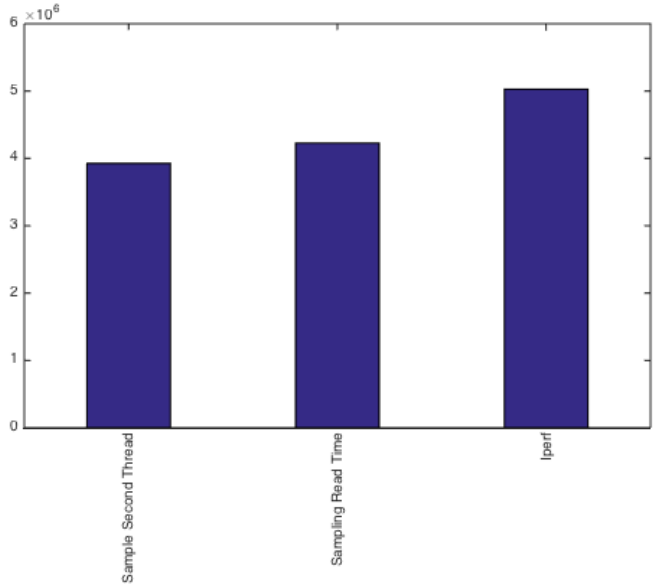

Figure 4.13: Transfered traffic uplink

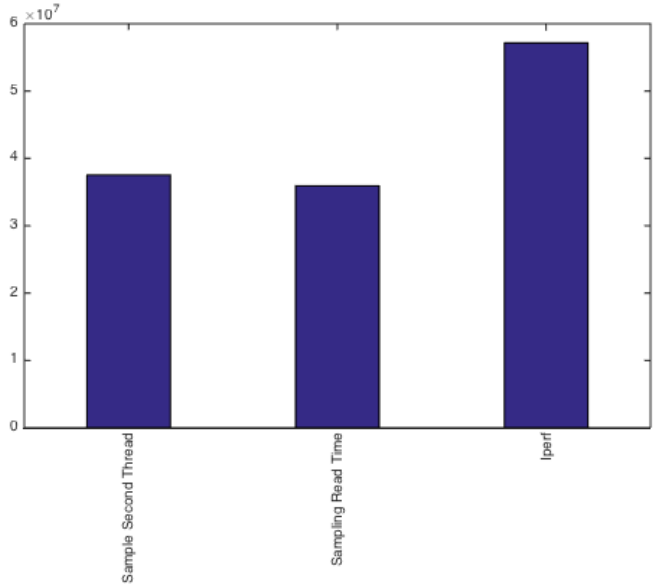

Figure 4.14: Transfered traffic downlink

4.5.3 Wi-Fi - GPON network

In order to verify the saturation point of the developed tool, the highest values of bandwidth tested were measured in a GPON network, where the upload and download rates are the same due to the transmission medium. A 100 Mbps GPON service over the optical fiber is used to connect the WiFi AP. Note that the measurements test setup used to perform this estimation were previously described in chapter three on sections 3.3.3 and 3.2.3, with the Nagle's algorithm turned on.

This network has an average RTT of 7.528 ms , being the route crossed by the packets the following:

1. dsldevice (192.168.1.254) 21.668 ms 1.930 ms 1.879 ms
2. 10.239.192.1 (10.239.192.1) 24.967 ms 6.524 ms 13.297 ms
3. bl3-77-5.dsl.telepac.pt (213.13.77.5) 4.871 ms 3.509 ms
4. bl3-77-6.dsl.telepac.pt (213.13.77.6) 5.128 ms 4.286 ms 3.651 ms
5. lis2-cr1-bu10-200.cprm.net (195.8.30.241) 7.506 ms 7.123 ms 5.648 ms
6. fccn.as1930.gigapix.pt (193.136.251.1) 7.211 ms 5.499 ms 4.578 ms
7. router60.10ge.cr1.lisboa.fccn.pt (193.137.0.14) 4.869 ms 5.128 ms 4.640 ms
8. 193.137.124.161 (193.137.124.161) 8.964 ms 6.430 ms

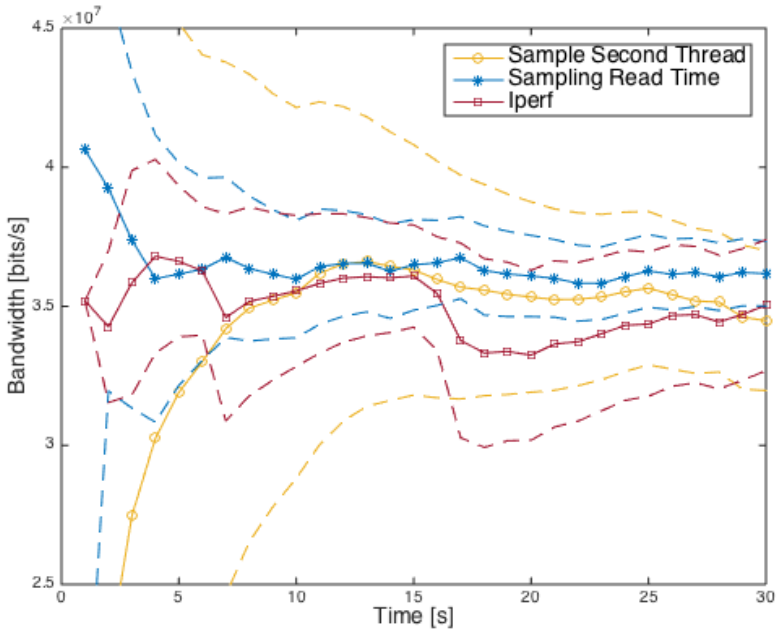

Figure 4.15: Uplink in GPON network

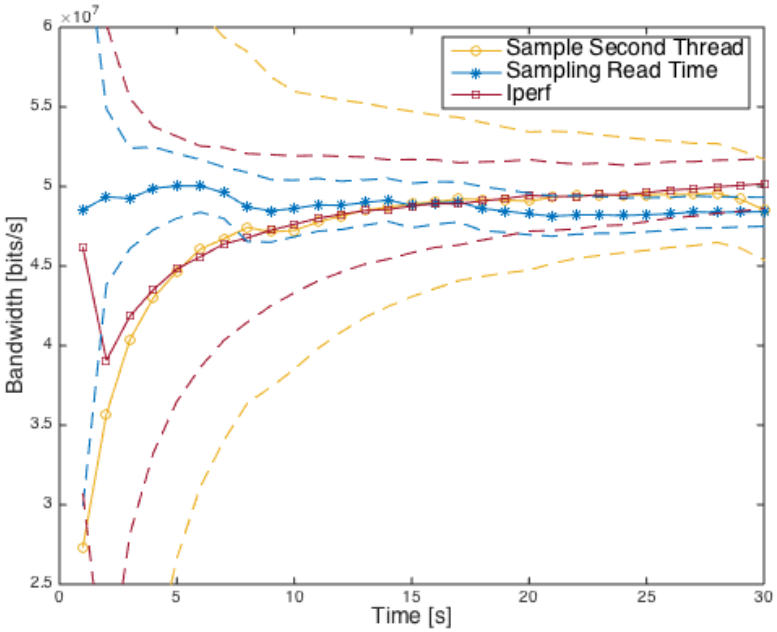

Figure 4.16: Downlink in GPON network

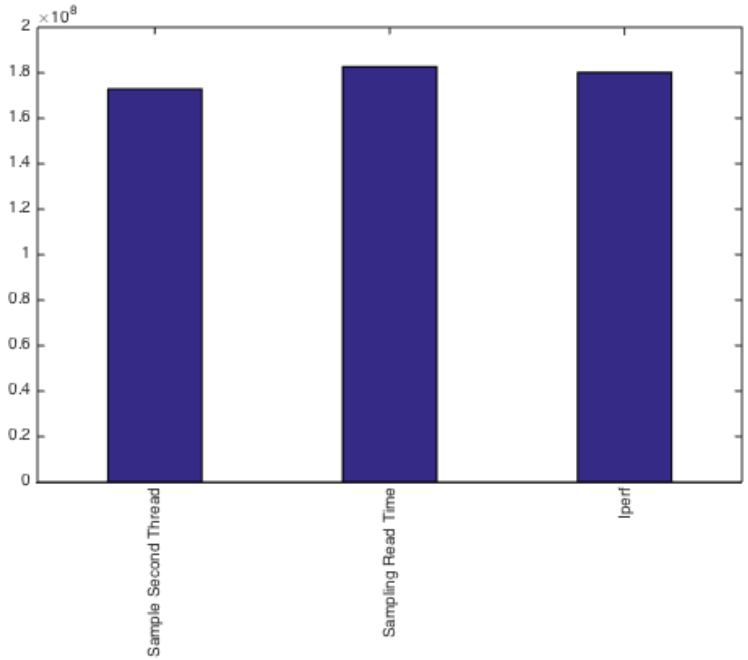

Figure 4.17: Transfered traffic uplink

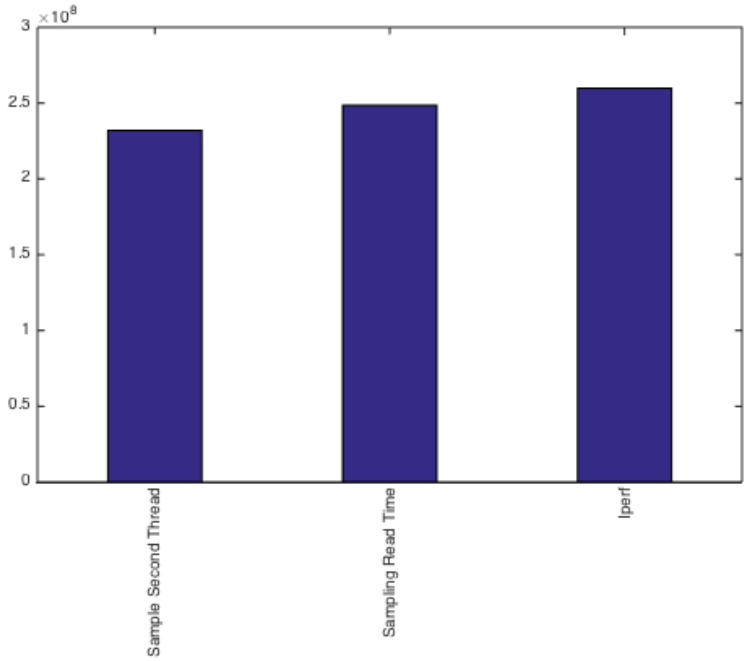

Figure 4.18: Transfered traffic downlink

As demonstrated by figures 4.15 and 4.16, the results measured using both methods of the application are valid, since there is an intersection between the error bounds of both, the iPerf and the deployed BTC algorithms.

The amount of bits sent over the network, illustrated in figures 4.17 and 4.18, accounts the traffic transmitted until reaching stable measurements, with the Jain's fairness index of the 5 slots higher than 0.999.

Once again, the Sample Second Thread seems to be the best algorithm between the deployed two, with an upload accuracy of 94.9959% and a download accuracy of 89.3068% whereas the Sampling Read Time obtained 97.2709% and 75.1540% respectively.

4.5.4 3G - Cellular network

As mentioned in section 4.5, the mobile measurements were performed for the three major Portuguese ISP, being presented here the results obtained from ISP Nº2 as an example of the 3G performance. Note that, the measurements were obtained in both directions with the Nagle's algorithm enabled.

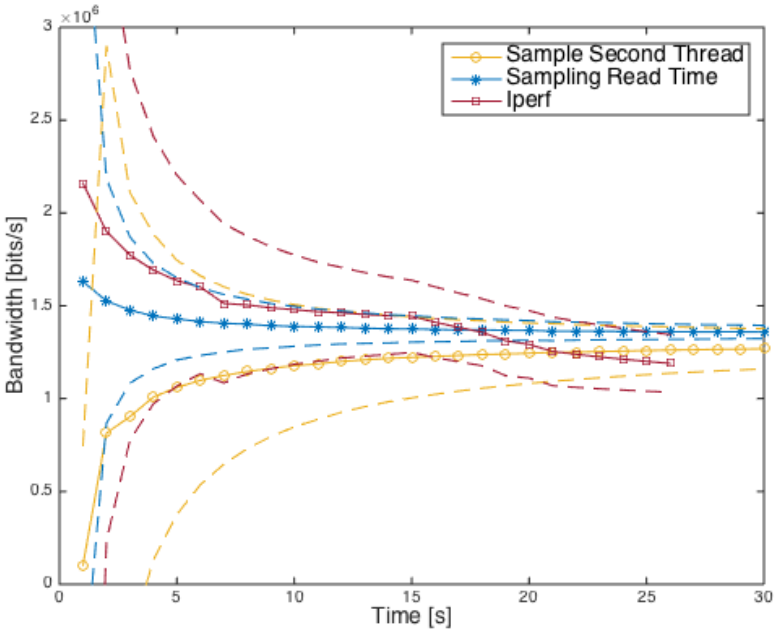

Figure 4.19: Upload in 3G network

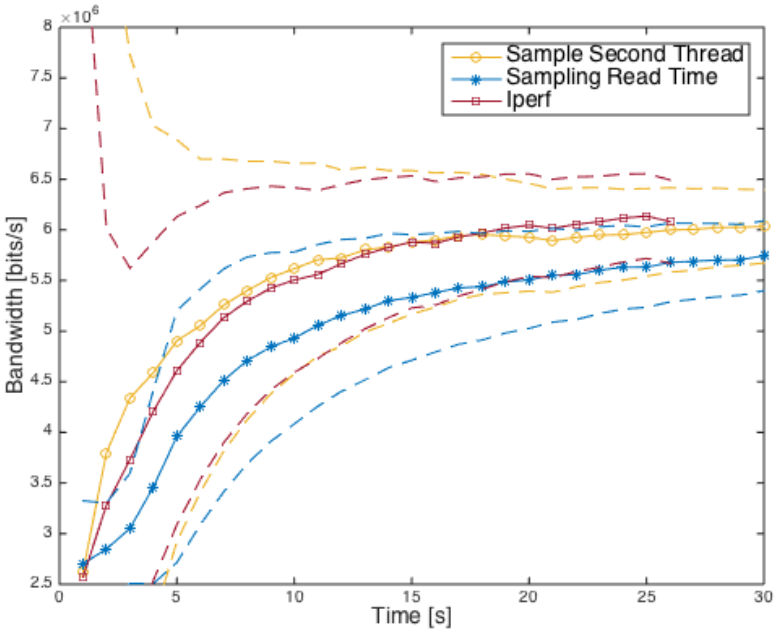

Figure 4.20: Download in 3G network

The results illustrated by the figure 4.19 were gathered by a mobile phone with the specifications defined in section 3.3.3. As illustrated by figures 4.19 and 4.20, there exists an intersection between the error bounds of both applications, Android and iPerf, which validates the results. On the uplink direction, the Sampling Read Time algorithm shows a lower standard deviation in relation to the mean value, when compared with the other

BTC algorithm and the iPerf tool. On the other half, the referred feature is not visible on the downlink direction, where both presented methods show a similar error bound.

Hence, the average percentage accuracy of the 5 stable records, is 75.3640% for the Sample Second Thread on the uplink and 97.8025% on the downlink direction. The Sampling Read Time have presented an accuracy of 94.2301% and 94.8488% for the upload and download respectively.

4.5.5 4G - Cellular network

Similarly to 3G, in order to have an example of the fourth generation technology, the 4G results shown in this section were performed at the ISP N°2 cellular network, using the same mobile phone described in the previous section.

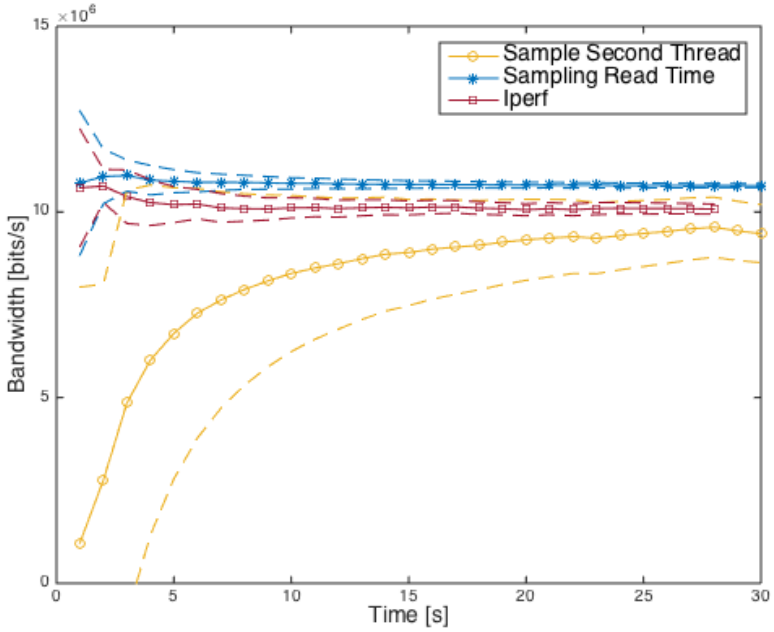

Figure 4.21: Upload in 4G network

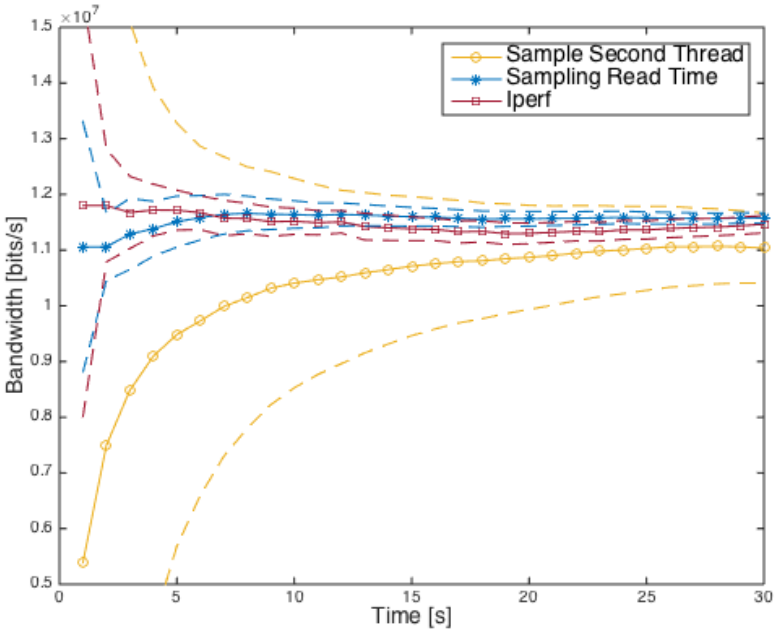

Figure 4.22: Download in 4G network

As expected, the throughput of the 4th generation of wireless mobile telecommunications technology is much higher than the one presented by 3G.

At the time of the measurements on ISP Nº2 cellular network, sometimes the tests were not performed over the defined 32 seconds, with the connection being cut after a time close to 30 seconds. This happened several times in different tests, which suggests that a traffic shapping mechanism is being applied by the ISP. Furthermore, a questioning fact about the iPerf settings comes up: the Nagle's algorithm seems to have no effect on the iPerf estimation in all the tests performed under the scope of this dissertation, which suggests that the iPerf3 implementation for Android called Magic iPerf [63], might be ignoring the command. Accordingly, the results, gathered with the Nagle turned on, validate the reliability of the Sample Second Thread method with an accuracy of 86.3612% on the downlink and also suggest the existence of limitations in the implementation of the problems related with the Sampling Read Time algorithm that presents accuracy of 66.5316%. However, on the uplink direction the latter have demonstrated a better performance with 95.1319% against 81.8085% presented by the Sample Second Thread.

4.6 3G Portuguese ISP benchmark

This section presents a benchmark evaluating the 3G performance of the three major Portuguese providers of Internet. The comparison was done only with the Nagle's algorithm turned on. The evaluation was performed using the Sample Second Thread algorithm, which is the one that has presented the better performance, and the iPerf tool. Moreover, the total bits injected on the cellular network are also shown.

It is important to refer that the tests evolving ISP Nº1 and ISP Nº3 were made on Lisbon city, in Parque das Nações, except for ISP Nº2 measurements that were performed at Saldanha due to a problem in the SIM card at the time of the first tests.

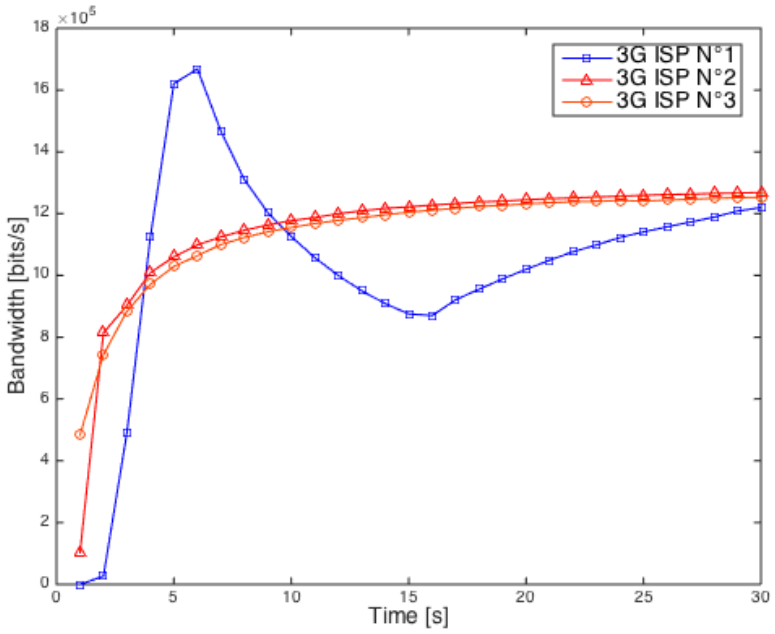

Figure 4.23: 3G benchmark uplink mean - Android

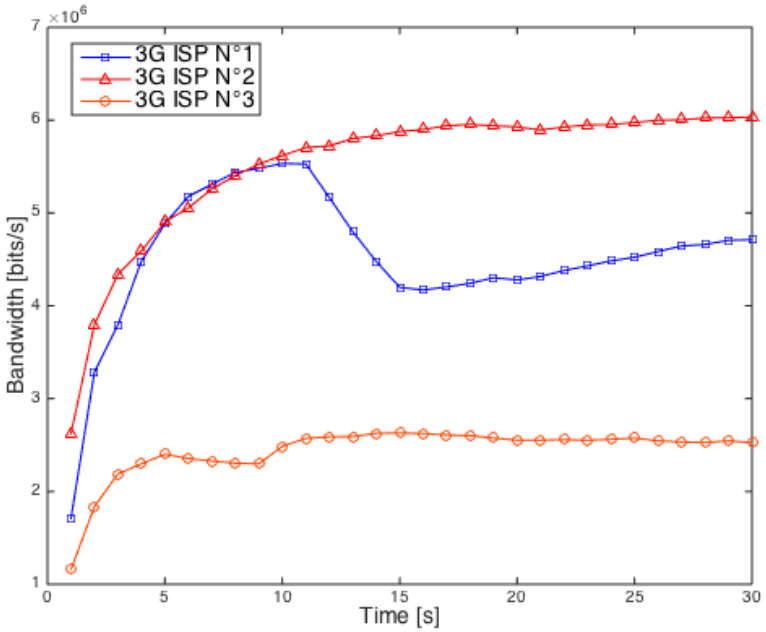

Figure 4.24: 3G benchmark downlink mean - Android

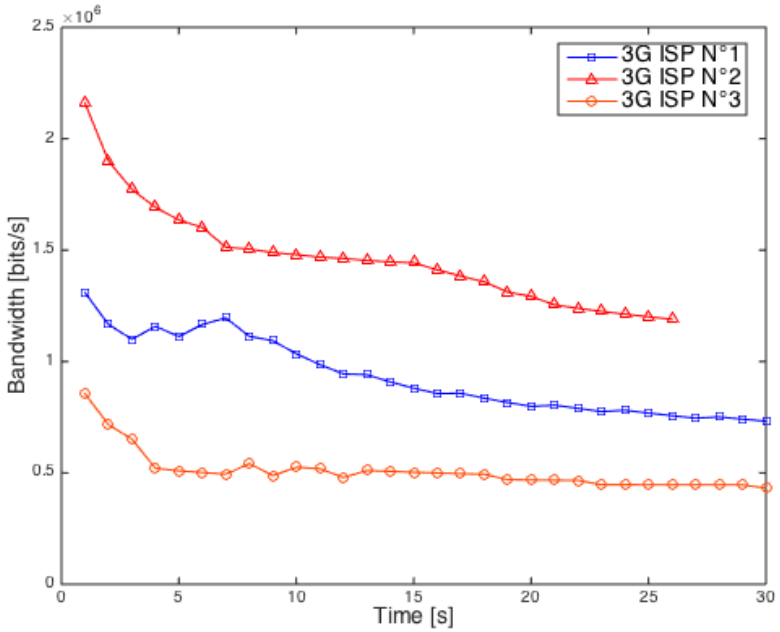

Figure 4.25: 3G benchmark uplink mean - Iperf

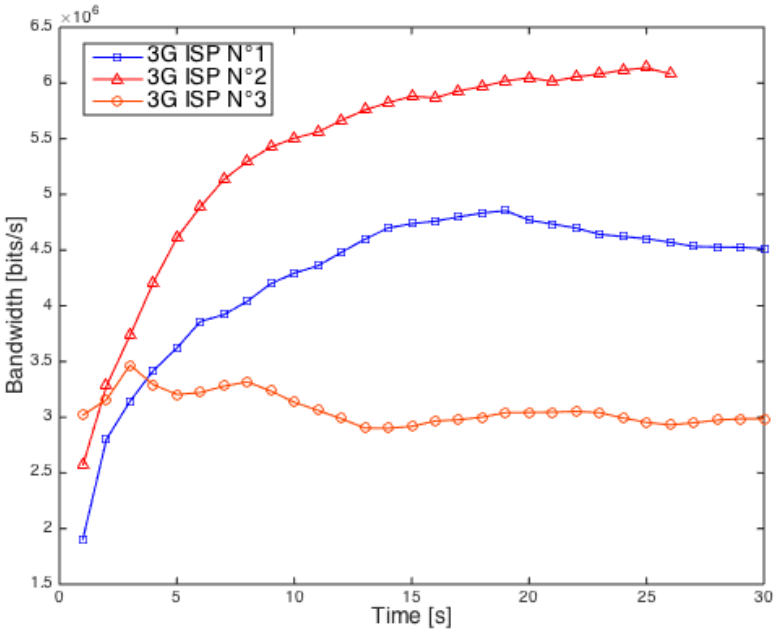

Figure 4.26: 3G benchmark downlink mean - Iperf

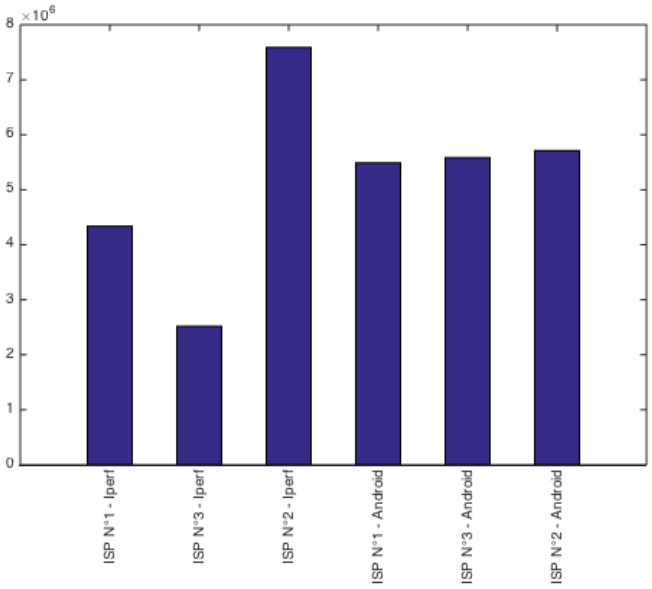

Figure 4.27: 3G transfered traffic on the uplink

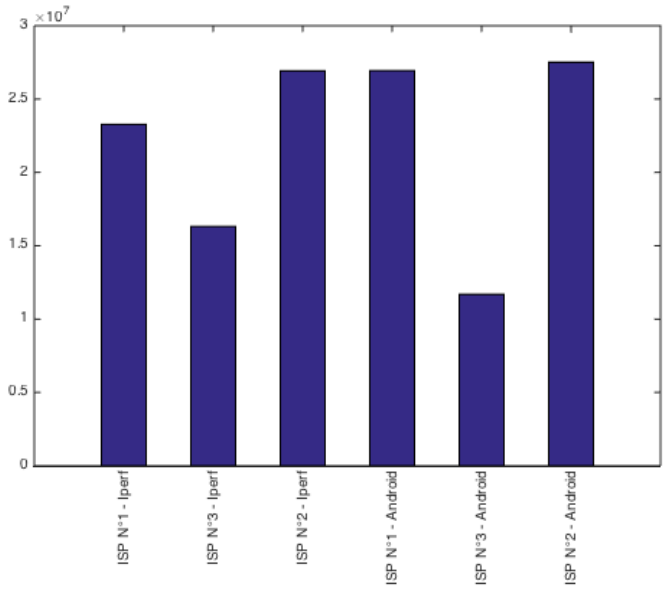

Figure 4.28: 3G transfered traffic on the downlink

Observing the plots, it is possible to verify that the Android application results are consistent when compared to the iPerf tool estimation. Note that ISP N°2, followed by ISP N°1 are the internet service providers that show the best overall performances, with the exception for figure 4.23 where the 3G ISP N°3 subplot is practically overlapping the 3G ISP N°2. Additionally, it is important to highlight the ISP N°1 traffic behaviour in figures 4.23 and 4.24, which suggest that a shapper is throttling the traffic when the throughput goes above a certain threshold.

The amount of traffic transferred in the tests is presented on figures 4.27 and 4.28. It depends on the homogeneity of the computed records. The sooner the Jain's fairness index is above 0.999, the lower is the number of bits sent over the network. In this case, the measurement results are consistent since ISP's with higher throughputs have injected more traffic.

4.7 4G Portuguese ISP benchmark

In the same conditions of the 3G benchmark described before, an evaluation of the 4G performance for the same three ISP is illustrated by figures between 4.29 and 4.32. At the end of this section it is shown the total bits transferred during the measurements. As in 3G scenario, the 4G benchmark were performed at the same locations.

By analysing the graphs, it is possible to verify that, as in the 3G scenario, ISP N°2 presents a better performance than the other two internet service providers.

The measurements obtained for the downlink by the Sample Second Thread algorithm and by the iPerf tool tend more or less to the same values, noticing that contrarily to 3G, the iPerf has a faster convergence than the deployed Android application.

Concerning the measurement of the total bits sent, similarly to the ADSL, GPON and 3G scenarios, the measurement considered the traffic transmitted until reaching a stable throughput.

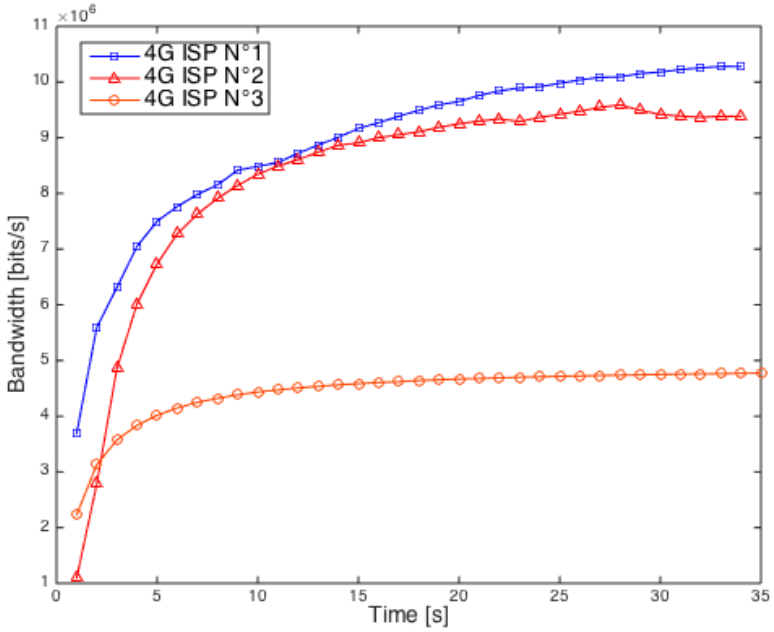

Figure 4.29: 4G benchmark uplink mean - Android application

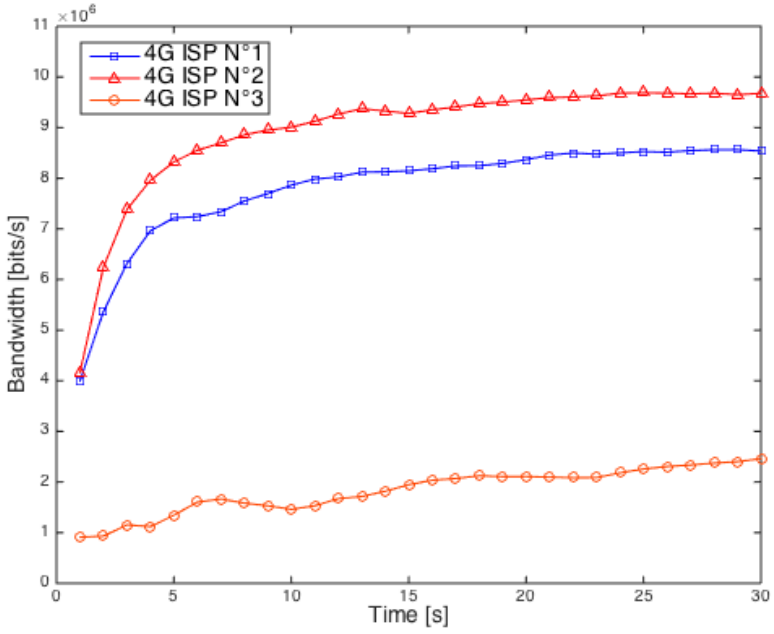

Figure 4.30: 4G benchmark downlink mean - Android application

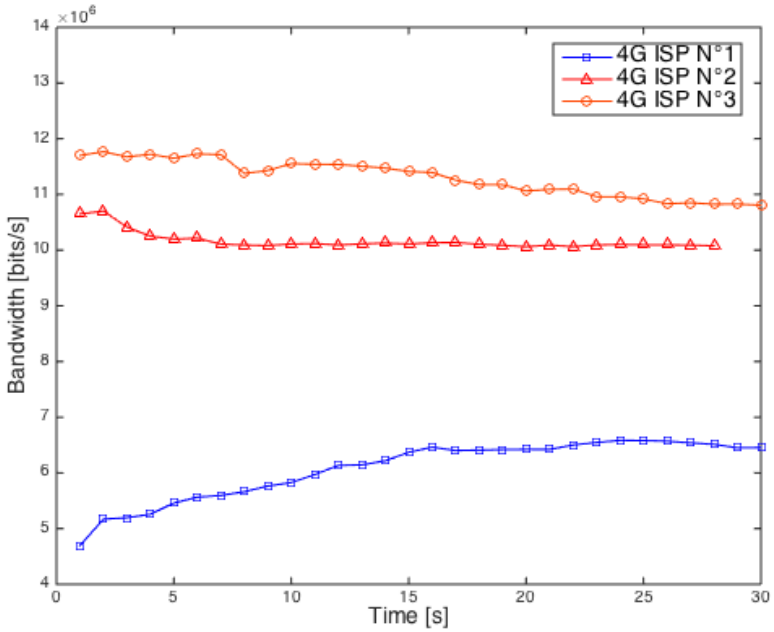

Figure 4.31: 4G benchmark uplink mean - Iperf tool

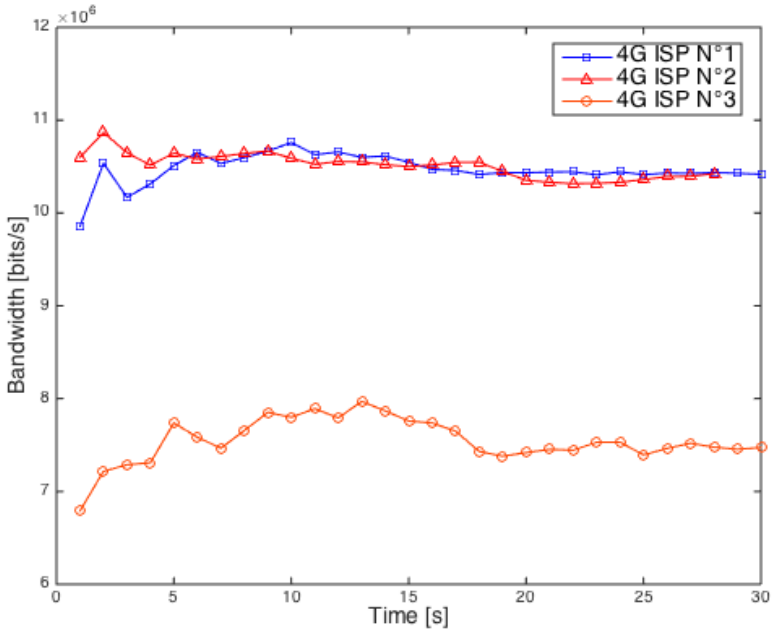

Figure 4.32: 4G benchmark downlink mean - Iperf tool

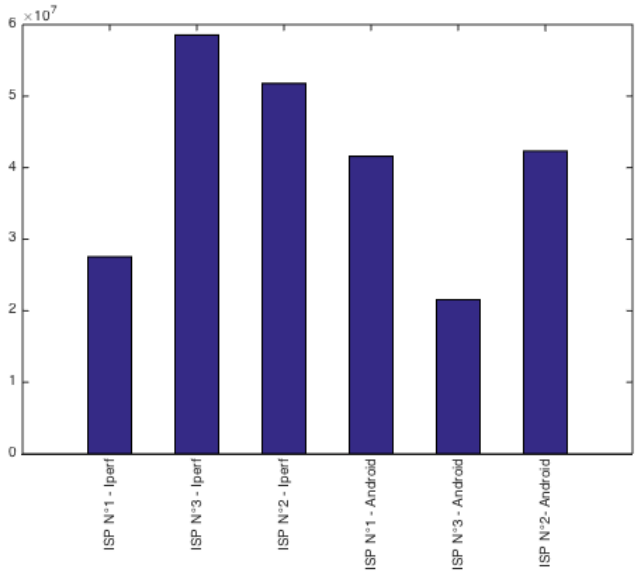

Figure 4.33: 4G transfered traffic on the uplink

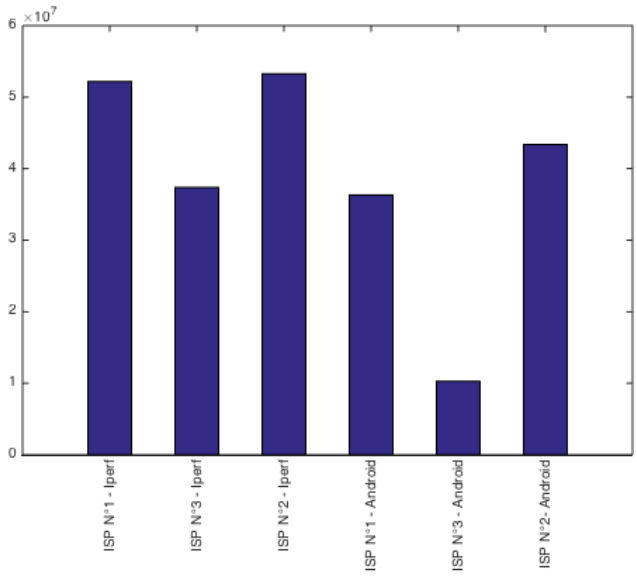

Figure 4.34: 4G transfered traffic on the downlink

It is important to refer that the measurement tests were performed according to the "QoS parameters and the corresponding measurement and evaluation procedures" documented in [64]. Simultaneously, the Ofcom specifies in [43] that the network capacity is measured by transmitting over a TCP channel at the maximum capacity for 30 seconds on the download and for 15 seconds on the uplink direction, being both of the connections interrupted after that. In this dissertation, a modification was proposed, based in the Jain's fairness index criteria, to stop measuring at the time the measurement samples are considered stable, which makes possible to inject less traffic into the network when the records present homogeneity.

4.8 Accuracy analysis

This section shows a summary table containing the accuracy obtained by the deployed BTC algorithms on both directions, in order to evaluate the performance of the developed methods in the studied network access scenarios upon comparison with the iPerf3 tool.

Table 4.1: Accuracy of BTC algorithms

Accuracy	Scenario	Algorithm			
		Upload	Download	Upload	Download
		Sample Second Thread		Sampling Read Time	
%	ADSL	78.1419	65.6999	80.9684	40.8521
	GPON	94.9959	89.3068	97.2709	75.154
	3G	75.364	97.8025	94.2301	94.8488
	4G	81.8085	86.3612	95.1319	66.5316
Average		82.577575	84.7926	91.900325	69.346625

As demonstrated by the previous table, the Sample Second Thread is more accurate, with an overall accuracy of up to 83%, while the Sampling Read Time only reaches 80%. Note that, the overall accuracy is the calculated mean of the average obtained for both directions. Therefore, in the download, the results present less accuracy than the opposite direction, which can be related to the concurrent traffic, that contributes to a more unstable connection when compared to the uplink.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Internet measurement studies have been relevant in the comprehension and analysis of the network performance, where metrology plays a major role through the interpretation of the metrics related with IP networks. Actively or passively, the measurement tools give an overview of the existing demands related to the enhancement of TCP scheme.

Accordingly, the client-server model has been widely used to support services in the field of distributed systems. Due to location transparency, it is possible to build a network architecture on which the resources are accessible without the user knowing the physical location of those resources. Hence, in the context of this dissertation, a client-server architecture was deployed to estimate the Bulk Transfer Capacity (BTC), which is defined as the maximum throughput achievable by a single TCP connection. This architecture is composed by a server awaiting for new requests to perform measurements, while the client, which can be a computer or a mobile phone, is the entity responsible for initiating the tests.

In the state of the art, presented in chapter two, several algorithms and tools to probe the Internet were described. In the scope of the developed work, two BTC estimation algorithms were successfully deployed, named Sample Second Thread and Sampling Read Time. As a conclusion of results analysis, the Sample Second Thread algorithm is the preferable method between the two, since it was the one that achieved the smallest divergence upon comparison with the iPerf3 tool, as shown in section 4.8. On the other half, the application deployed on the developed work was dependent from the Java accuracy clock, which depends to the underlying operating system (OS), being the measured time unit in the tens of milliseconds depending on the OS. By that, in some cases, a small difference in measurement results is denoted, when compared to the iPerf tool. This

phenomenon is described in the section 4.5.2, where a comparison between the deployed BTC algorithms and the iPerf tool is performed.

Despite of the algorithms used to perform network probing, the throughput of a network, which is expressed by the the equation 4.1, is directly related with the TCP congestion control algorithm defined by the operating system. Since the TCP windows are decisive factors to throughput, limiting the size of those buffers means in practice that an upper bound is being imposed to the amount of data that can transit over a network. The Nagle's algorithm, which is enabled by default in the majority of the systems, was deployed on TCP/IP networks as a strategy to improve TCP efficiency. Nagle combines a number of outgoing packages, sending them as a burst. When it is disabled, the transition from slow start period to congestion avoidance phase on the congestion control mechanism is more denoted, as illustrates figure 4.9.

Another factor that might affect the TCP performance are the traffic shapping techniques, which involve regulating the flow of packets in order to trace a traffic profile for the client. This bandwidth throttling scheme was felt on the ISP Nº2 measurements over the cellular network, where the connection was cut after a certain time transmitting at the maximum capacity, as explained on the section 4.5.1.

When a packet is sent from A to B, there are several possible bottlenecks over the path. It was noticed that the accuracy of the developed algorithms is lower on the downlink direction than on the uplink, which can be explained by the concurrent traffic. The measurements tests performed over the GPON service are more accurate and have less dispersion when compared to other scenarios, which demonstrates the influence of traffic asymmetry. Moreover, the maximum bandwidth estimation measured by the deployed tool was of 50 Mbps, which corresponds to the iPerf3 tool estimation.

In a nutshell, the proposed method is able to measure values of throughput in line with the ones measured by the Ofcom method, sending less data.

5.2 Future work

As a future for the developed work, a Kalman filter may be applied to the results in order to measure the available bandwidth, despite a strong nonlinearity in the system measurement model. The Bandwidth Available in Real-Time (BART) [65], presents a reasonable accuracy with little computational efforts and lower extra traffic load injected on the network from the probe packets.

After this first development, network emulation can be seen as the next objective, since other features, such as restricting the bandwidth or increasing the delay can be applied, originating new results which can conduct to a new study.

Another techniques to estimate throughputs might be developed, using for example several concurrent TCP flows, which will allow to study the gain of each individual flow, from N to $N+1$ connections.

Finally, the GUI can be upgraded on both environments, Java and Android, providing a better user experience.

Bibliography

[1] D. E. C. E. R. Laboratory, R. Jain, D. Chiu, and W. Hawe. *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System*. Eastern Research Laboratory, Digital Equipment Corporation, 1984. URL: https://books.google.pt/books?id=M2QLGwAACAAJ.

[2] R Carter. "Measuring bottleneck link speed in packet-switched networks". In: *Performance Evaluation* 27-28.1 (1996), pp. 297–318. DOI: 10.1016/0166-5316(96)00036-3.

[3] R Prasad and Others. "Bandwidth estimation: metrics, measurement techniques, and tools". In: *IEEE Network* 17.6 (2003), pp. 27–35.

[4] J. Strauss. "Choosing Internet Paths with High Bulk Transfer Capacity". MA thesis. Massachusetts Institute of Technology, Sept. 2002.

[5] F. Michaut and F. Lepage. "Application-oriented Network Metrology: Metrics and Active Measurement Tools". In: *IEEE Communications Surveys & Tutorials* 7.2 (Apr. 2005), pp. 2–24.

[6] V Paxson. "Measurements and analysis of end-to-end Internet dynamics". In: *University of California at Berkeley* (1998). DOI: 10.2172/551971.

[7] G. Almes, S. Kalidindi, and M. Zekauskas. *"A One-way Delay Metric for IPPM"*. RFC 2679. Fremont, CA, USA, 1999.

[8] S. Shalunov and *et al.* *"A One-Way Active Measurement Protocol (OWAMP)"*. RFC 4656. 2006.

[9] R. Koodli and R. Ravikanth. *"One-way Loss Pattern Sample Metrics"*. RFC 3357. 2002.

[10] J. Bellardo and S. Savage. "Measuring packet reordering". In: *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2002). DOI: 10.1145/637215.637216.

[11] J. Nagle. *"Congestion Control in IP/TCP Internetworks"*. RFC 896. Jan. 1984.

[12] N. Hu and P. Steenkiste. "Evaluation and characterization of available bandwidth probing techniques". In: *IEEE Journal on Selected Areas in Communications* 21.6 (2003), pp. 879–894. DOI: 10.1109/JSAC.2003.814505.

[13] J. Strauss, D. Katabi, and F. Kaashoek. "A measurement study of available bandwidth estimation tools". In: *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (2003), pp. 39–44. DOI: http://doi.acm.org/10.1145/948205.948211.

[14] C. Dovrolis, P. Ramanathan, and D. Moore. "What do packet dispersion techniques measure?" In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)* 2 (2001), pp. 905–914. DOI: 10.1109/INFCOM.2001.916282.

[15] M. Jain and C. Dovrolis. "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput". In: *IEEE/ACM Transactions on Networking* 11.4 (Aug. 2003), pp. 537–549. DOI: 10.1109/TNET.2003.815304.

[16] M. Jain and C. Dovrolis. "Pathload: A Measurement Tool for End-to-End Available Bandwidth". In: *In Proceedings of Passive and Active Measurements (PAM) Workshop*. 2002, pp. 14–25.

[17] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. *pathChirp: Efficient Available Bandwidth Estimation for Network Paths*. 2003.

[18] S Saroiu, P. K. Gummadi, and S. D. Gribble. "SProbe: A Fast Technique for Measuring Bottleneck Bandwidth in Uncooperative Environments". In: *Procedings of IEEE Infocom* (2002), pp. 1–11.

[19] K. Lai and M. Baker. "Nettimer: A Tool for Measuring Bottleneck Link Bandwidth". In: Proc. USENIX Symp. Internet Tech. and Sys. San Francisco, USA, Mar. 2001, pp. 123–134.

[20] V. Jacobson. "Pathchar — a Tool to Infer Characteristics of Internet Paths". In: 1997.

[21] P. Beyssac. *Bing: A point-to-point bandwidth measurement tool based on PING*. 1995.

[22] A. Downey. "Using pathchar to estimate Internet link characteristics". In: *In Proceedings of ACM SIGCOMM*. 1999, pp. 241–250.

[23] K. Lai and M. Baker. "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay". In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '00. Stockholm, Sweden: ACM, 2000, pp. 283–294. DOI: 10.1145/347059.347557.

[24] V. Bajpai and J. Schönwälder. "A Survey on Internet Performance Measurement Platforms and Related Standardization Efforts". In: *IEEE Communications Surveys Tutorials* 17.3 (2015), pp. 1313–1341. DOI: 10.1109/COMST.2015.2418435.

[25] U. Goel, M. P. Wittie, K. C. Claffy, and A. Le. "Survey of End-to-End Mobile Network Measurement Testbeds, Tools, and Services". In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 105–123. DOI: 10.1109/COMST.2015.2485979.

[26] S. Sonntag, J. Manner, and L. Schulte. "Netradar - Measuring the wireless world". In: *Modeling Optimization in Mobile, Ad Hoc Wireless Networks (WiOpt), 2013 11th International Symposium on*. May 2013, pp. 29–34.

[27] E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio. "Sensing the Internet through crowdsourcing". In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*. Mar. 2013, pp. 248–254. DOI: 10.1109/PerComW.2013.6529490.

[28] A. Faggiani, E. Gregori, L. Lenzini, S. Mainardi, and A. Vecchio. "On the feasibility of measuring the internet through smartphone-based crowdsourcing". In: *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2012 10th International Symposium on*. May 2012, pp. 318–323.

[29] F. Disperati, D. Grassini, E. Gregori, A. Improta, L. Lenzini, D. Pellegrino, and N. Redini. "SmartProbe: A Bottleneck Capacity Estimation Tool for Smartphones". In: *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. Aug. 2013, pp. 1980–1985. DOI: 10.1109/GreenCom-iThings-CPSCom.2013.371.

[30] B. Augustin, T. Friedman, and R. Teixeira. "Measuring Load-balanced Paths in the Internet". In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC '07. San Diego, California, USA: ACM, 2007, pp. 149–160. DOI: 10.1145/1298306.1298329.

[31] S. McKee, A. Lake, P. Laurens, H. Severini, T. Wlodek, S. Wolff, and J. Zurawski. "Monitoring the US ATLAS Network Infrastructure with perfSONAR-PS". In: *Journal of Physics: Conference Series* 396.4 (2012), p. 042038.

[32] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. "Broadband Internet Performance: A View from the Gateway". In: *Proceedings of the ACM SIGCOMM 2011 Conference*. SIGCOMM '11. Toronto, Ontario, Canada: ACM, 2011, pp. 134–145. DOI: 10.1145/2018436.2018452.

[33] S. Sundaresan, S. Burnett, N. Feamster, and W. de Donato. "BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, June 2014, pp. 383–394.

[34] M. A. Sánchez, J. S. Otto, Z. S. Bischof, and F. E. Bustamante. "Dasu - ISP Characterization from the Edge: A BitTorrent Implementation". In: *Proceedings of the ACM SIGCOMM 2011 Conference*. SIGCOMM '11. Toronto, Ontario, Canada: ACM, 2011, pp. 454–455. DOI: 10.1145/2018436.2018517.

[35] M. Rimondini, C. Squarcella, and G. D. Battista. "Towards an Automated Investigation of the Impact of BGP Routing Changes on Network Delay Variations". In: *PAM*. 2014.

[36] U. Goel, A. Miyyapuram, M. P. Wittie, and Q. Yang. "MITATE: Mobile internet testbed for application traffic experimentation". In: *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST* 131 (2014), pp. 224–236. DOI: 10.1007/978-3-319-11569-6_18.

[37] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen. "PhoneLab: A large programmable smartphone testbed". In: *Proceedings of First International Workshop on Sensing and Big Data Mining* (2013), pp. 1–6. DOI: 10.1145/2536714.2536718.

[38] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. "Measuring Home Broadband Performance". In: *Commun. ACM* 55.11 (Nov. 2012), pp. 100–109. DOI: 10.1145/2366316.2366337.

[39] G. Developers. "*Google Maps Developer Documentation*". Mar. 2015. URL: https://developers.google.com/maps/documentation/ (visited on 06/02/2016).

[40] Vodafone. "*Take control of your data and Wi-Fi usage - Vodafone NetPerform*". Mar. 2015. URL: http://www.vodafone.co.uk/discover-vodafone/apps-and-downloads/vodafone/ (visited on 06/02/2016).

[41] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. "Netalyzr: Illuminating the Edge Network". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, pp. 246–259. DOI: 10.1145/1879141.1879173.

[42] G. Malkin. "*Traceroute Using an IP Option*". RFC 1393. Jan. 1993.

[43] "*Measuring mobile broadband performance in the UK*". 4G and 3G network performance. Nov. 2014.

[44] iPerf. *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. URL: https://iperf.fr/iperf-doc.php (visited on 03/12/2017).

[45] Yatta Solutions GmbH. *UML Lab*. URL: http://www.uml-lab.com/en/uml-lab/ (visited on 03/11/2017).

[46] Apache Commons. *Apache Commons Math 3.6 API*. URL: http://commons.apache.org/proper/commons-math/javadocs/api-3.6/index.html (visited on 03/14/2017).

[47] teamwork523. *MobiPerf's prototype on bandwidth testing using packet train*. URL: https://github.com/teamwork523/MobiBand (visited on 03/10/2017).

[48] Oracle. *Java Platform, Standard Edition 7, API Specification*. URL: https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html (visited on 03/10/2017).

[49] Oracle. *Java Platform, Standard Edition 7, API Specification*. URL: https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html (visited on 03/10/2017).

[50] MathWorks. *MATLAB version 8.4.0.150421 (R2014b)*. Natick, Massachusetts, 2014. URL: https://www.mathworks.com/products/matlab.html (visited on 02/27/2017).

[51] W. Falkena. *MathWorks*. Aug. 2010. URL: https://www.mathworks.com/matlabcentral/fileexchange/28518-xml2struct (visited on 02/27/2017).

[52] Open Handset Alliance. *Industry Leaders Announce Open Platform for Mobile Devices*. URL: http://www.openhandsetalliance.com/press_110507.html (visited on 03/11/2017).

[53] Developers Android. *Platform Architecture*. URL: https://developer.android.com/guide/platform/index.html#api-framework (visited on 03/11/2017).

[54] B. Phillips, C. Stewart, B. Hardy, and K. Marsicano. *Android Programming: The Big Nerd Ranch Guide*. 2nd. Big Nerd Ranch, 2015.

[55] J. Gehring. *Android Graph Library for creating zoomable and scrollable line and bar graphs*. URL: https://github.com/appsthatmatter/GraphView (visited on 03/12/2017).

[56] Dafontfree. *Eurostile Regular Font*. URL: http://www.dafontfree.com/eurostile-regular-font (visited on 03/12/2017).

[57] M. Mathis and M. Allman. *"A Framework for Defining Empirical Bulk Transfer Capacity Metrics"*. RFC 3148. July 2001.

[58] A. Tanenbaum and D. Wetherall. *Computer Networks*. Pearson Prentice Hall, 2011. URL: https://books.google.ca/books?id=I764bwAACAAJ.

[59] G. Snedecor and W. Cochran. *Statistical methods*. Iowa State University Press, 1980. URL: https://books.google.pt/books?id=x4EuAAAAIAAJ.

[60] Wireshark. *Network Protocol Analyzer version 2.2.1*. URL: https://www.wireshark.org/ (visited on 03/16/2017).

[61] Gordon Lyon. *Nping*. URL: https://nmap.org/book/nping-man-tcp-mode.html (visited on 03/14/2017).

[62] Oracle. *Java Platform, Standard Edition 7, API Specification*. URL: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#currentTimeMillis() (visited on 02/27/2017).

[63] NextDoorDeveloper. *Magic iPerf including iPerf3*. URL: https://play.google.com/store/apps/details?id=com.nextdoordeveloper.miperf.miperf&hl=pt_PT (visited on 03/21/2017).

[64] *"Speech and multimedia Transmission Quality (STQ); QoS Parameter Measurements based on fixed Data Transfer Times"*. Technical Report TR 102 678 V1.2.1. May 2011.

[65] S. Ekelin, M. Nilsson, E. Hartikainen, A. Johnsson, J.-E. Mångs, B. Melander, and M. Björkman. "Real-time Measurement of End-to-End Available Bandwidth Using Kalman Filtering". In: *In proceedings to the Network Operations and Management Symposium*. Will soon be published here. 2006. URL: http://www.es.mdh.se/publications/897-.