UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Facoltà di Ingegneria

Corso di Dottorato di Ricerca in Ingegneria Informatica ed Automatica
XXIV Ciclo
Dipartimento di Informatica e Sistemistica

Large Scale Benchmarking of Broadband Access Networks: Issues, Methodologies, and Solutions

Walter de Donato

Ph.D. Thesis

Tutor
Prof. Antonio Pescapé

Cotutor
Prof. Giorgio Ventre
Prof. Nick Feamster

Coordinator
Prof. Francesco Garofalo

November 2011

To my wife Valeria...
Her love keeps me
going on every day!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we introduce the scenario in which this study is conducted, outlining the motivations stimulating our research work. Afterwards, we briefly describe the contributions provided by this thesis with respect to the literature. The ending part of the chapter outlines the organization of the thesis.

1.1 The Internet scenario

The Internet today counts about 2.1 billion users worldwide, among which 580 million are residential broadband subscribers [1]. As reported in Fig. 1.1, these numbers are growing rapidly and, with the diffusion of broadband access on mobile terminals[1], this process is accelerating further.

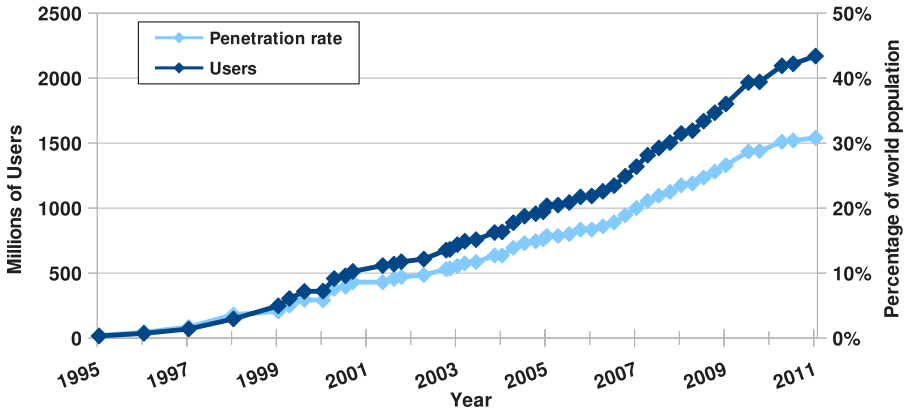


Figure 1.1: Internet users growth and penetration rate.

[1] 5.1 billion mobile subscribers were reported in 2010 [2]

Most people rely on Internet connectivity for everyday activities, making broadband access an essential resource, whose performance have not been widely studied in literature.

On one side, it is a matter of fact that users frequently experience performance problems, which can be due to causes both internal and external to their network. In the first case, the user himself can be responsible by not properly configuring his computer or the devices in his network (e.g. router, access point, set-top-box, ...). In the second case, it may depend on the access link - bad conditions of wiring (e.g. old wires, low signal/noise ratio, long distance from the telephone central, ...) or environment (e.g. electromagnetic interference) - or on the policies adopted by the Internet Service Provider (ISP). Identifying the root cause in this context is not trivial, due to the heterogeneity of available access technologies, both wired (e.g. DSL, Cable, Fiber, ...) and wireless (e.g. WiFi, WiMAX, GPRS, UMTS, ...).

On the other side, broadband access networks performance is made crucial by the diffusion of new generation applications and services, progressively providing - through a single interface - more interactions among the users and between the users and the network. This is promoting the development of *multi-channel applications* (e.g. Skype, Facebook, GMail, ...) that are specifically designed to transparently manage different services delivered on different channels, providing a single access point for the users. Therefore, in order to guarantee a satisfying Quality of Experience (QoE), the access network has to satisfy tighter and tighter requirements. However, managing the Quality of Service (QoS) in presence of such applications is even more difficult, because it would be necessary to treat differently each communication channel depending on its typology.

Benchmarking performance of broadband access networks, however, is not as simple as running one-time "speed tests". There exist countless tools to measure Internet performance [3, 4, 5, 6]. Previous work has studied the typical download and upload rates of home access networks [7, 8]; others have found that modems often have large buffers [8], and that DSL links often have high latency [9]. These studies have shed some light on access-link performance, but they have typically run one-time measurements either from an end-host inside the home (from the "inside out") or from a server on the wide-area Internet (from the "outside in"). Because these tools run from end-hosts, they cannot analyze the effects of confounding factors such as home network cross-traffic, the wireless network, or end-host configuration. Also, many of these tools run as one-time measurements. Without continual measurements of the same access link, these tools cannot

establish a baseline performance level or observe how performance varies over time.

Accordingly, many organizations worldwide have been recently active in the development of methodologies and in the definition of metrics to evaluate the performance offered by ISPs. Regulators (e.g. as the Federal Communication Commission (FCC) in the USA [10, 11, 12], the Office of Communications (OFcom) in the UK [13], and the Authority for Communications Guarantees (AGCOM) in Italy [14], ...), policymakers (e.g. the European Community [15]), and independent organizations (e.g. DSL Forum [16]) are actively developing performance-testing metrics for access providers.

1.1.1 Heterogeneity of access technologies

Today people can choose among many different technologies to access the Internet, which can be divided in two main categories: wired and wireless. In the following paragraphs we propose a brief overview of the most widely deployed access technologies in order to highlight their complexity and heterogeneity.

Wired technologies

Among all the wired access technologies, the most common rely on preexisting infrastructures to exploit the evident economic advantage. Digital Subscriber Line (DSL) reuse preexisting telephone wires working on higher frequency bands, while Cable reuse existing coaxial cables deployed by Cable Television (CATV) providers. Apart from those solutions the only other technology which is resulting progressively successful is based on optical fibers, which at a higher cost provide much superior performance.

DSL access networks. DSL access networks are based on preexisting telephone lines, thus relying on an unshielded twisted copper pair wire. Subscribers have dedicated lines between their own DSL modems and the closest DSL Access Multiplexer (DSLAM). The DSLAM multiplexes data between the access modems and upstream networks, as shown in Figure 1.2. The physical connection between a customer's home and the DSLAM is often referred to as the *last mile*. The most common type of DSL access is asymmetric (ADSL), which provides different upstream and downstream rates. The ITU-T standardization body establishes that the achievable rate for ADSL 1 [17] is 12 $Mbps$ downstream and 1.8 $Mbps$ upstream. The ADSL2+ specification [18] extends the capacity of ADSL

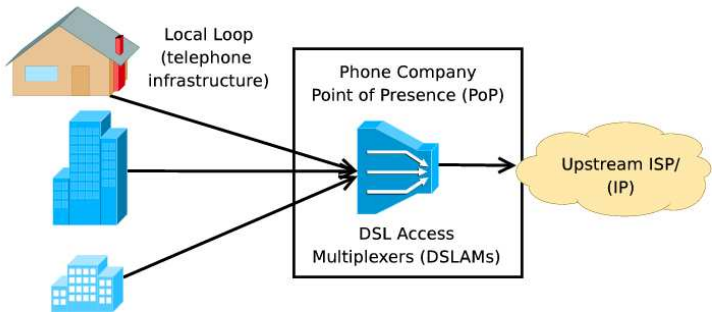


Figure 1.2: DSL access network.

links to at most 24 *Mbps* downstream and 3.5 *Mbps* upstream, and some non-standard implementations claim to reach downstream rates up to 28 *Mbps* (e.g. Free ISP in France). Although the ADSL technology is theoretically able to reach these speeds, there are many factors that limit the capacity in practice. An ADSL modem negotiates the operational rate with the DSLAM (often called the *sync rate*); this rate depends on the quality of the last mile, which is mainly determined by the distance to the DSLAM from the user's home, bad wiring conditions, and noise on the line. For example, ADSL 1 can reach up to 6 *Mbps* with a 4 *km* distance to the DSLAM.

The most advanced standard of DSL is represented by the Very-high-speed DSL (VDSL), which is designed to support the wide deployment of triple play services such as voice, video, data, high definition television (HDTV) and interactive gaming. ITU G.993.1 standard defines its first release, which is able to reach up to 52 *Mbps* downstream and 16 *Mbps* upstream rates on a single twisted copper pair wire. ITU G.993.2 standard defines its enhanced version (VDSL2), which supports the transmission of both asymmetric and symmetric aggregate data rates up to 200*Mbps* on twisted pairs using a bandwidth up to 30 MHz. The main limitation of such standard consists in the quick performance deterioration from a theoretical maximum of 250 *Mbps* at source, to 100 *Mbps* at 500 *m* and 50 *Mbps* at 1 *km*.

The best service plan that a DSL provider advertises usually represents the rate that customers can achieve at IP layer if they have a good connection to the DSLAM. It is important to notice that the maximum link capacity at IP layer is lower than the sync rate because of the overhead of underlying protocols. Providers also offer service plans with lower rates and can rate-limit customers traffic at the DSLAM (e.g. using weighted fair queuing). Modem configuration can also affect performance. DSL users or providers

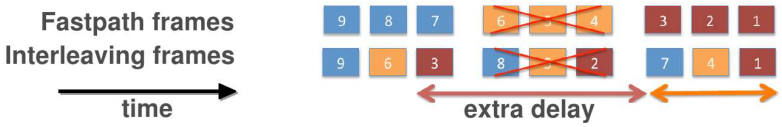


Figure 1.3: DSL fastpath vs interleaving.

configure their modems to operate in either *fastpath* or *interleaved* mode. In *fastpath* mode, data is exchanged between the DSL modem and the DSLAM in the same order as they are received, which minimizes latency by preventing error correction from being applied across frames. Thus, ISPs typically configure fastpath only if the line has a low bit error rate. *Interleaving* increases robustness to line noise at the cost of increased latency by splitting data from each frame into multiple segments and interleaving those segments with one another before transmitting them. For example, in a batch of frames (see Fig. 1.3), the first frame would contain the first segment from each frame in the original data stream, the second frame would contain the second, and so on. This mechanism allows the receiver to apply Forward Error Correction (FEC) techiques to recover from bursty line errors.

Cable access networks. In cable access networks, groups of users send data over a shared medium (typically a coaxial cable). At a regional *headend*, a Cable Modem Termination System (CMTS) receives these signals and converts them to Ethernet, as shown in Figure 1.4. The physical connection between a customer's home and the CMTS is often referred to as the *local loop*. Users buy a service plan from a provider that typically offers a *maximum* capacity in both the upstream and downstream directions. In cable networks, the most widely deployed version of the standard is Data Over Cable Service Interface Specification version 2 (DOCSIS 2.0) [19], which specifies downstream rates up to 42.88 *Mbps* and upstream rates up to 30.72 *Mbps* in the United States. The latest standard, DOCSIS 3.0, allows for hundreds of megabits per second by bundling multiple channels. Cable providers often offer service plans with lower rates and shape each user's traffic to enforce fairness on the local loop usage. The service plan rate limit is configured at the cable modem and is typically implemented using a token bucket rate shaper. Many cable providers also apply *PowerBoost*^{TM} , which allows users to download (and, in some cases, upload) at rates that are higher than the contracted ones, for an initial part of a transfer. The actual rate that a cable user receives will vary with the network utilization of

other users connecting to the same headend. The CMTS controls the rate at which cable modems transmit. For instance, Comcast describes that when a CMTS's port becomes congested, it ensures fairness by scheduling heavy users on a lower priority queue [20].

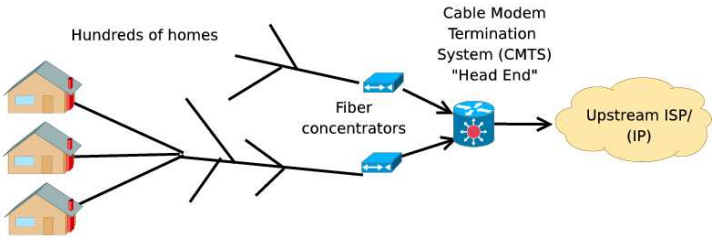


Figure 1.4: Cable access network.

Optical access networks. A single strand of fiber offers a total bandwidth of 25 THz, potential far in excess of the wireless and any other known transmission medium. More important, optical networks lend themselves well to offloading electronic equipment by means of optical bypassing, as well as reducing their complexity, footprint, and power consumption significantly.

Basically, three architectures may be deployed for the fiber access network (see Fig. 1.5) [21]. *Point-to-point* architectures (Fig. 1.5a) provide individual fibers from the local exchange to each home. Many fibers are needed, which entails high first installation costs, but also provides the ultimate capacity and the most flexibility to upgrade services for customers individually. In the local exchange, as many fiber terminals are needed as there are homes, so floor space and powering may become issues. *Active star* architectures (Fig. 1.5b) provide a single fiber which carries all the traffic to an active node close to the end users, from where individual fibers run to each cabinet/home/building. Only a single feeder fiber is needed, and a number of short branching fibers to the end users, which reduces costs; but the active node needs powering and maintenance. It also needs to withstand a wider range of temperatures than in-door equipment. The active node may be located in a cabinet at the street curb site (fiber to the cabinet or FTTC), or in the basement of a multi-dwelling units building (fiber to the building or FTTB) from where the communication traffic is run throughout the building by copper wired and wireless local area networks at speeds higher than 100 *Mbps*. *Passive star* architectures (Fig. 1.5c) provide a passive optical power splitter/combiner that feeds the individual short branching fibers to the end users. In addition to the reduced installation costs of a single

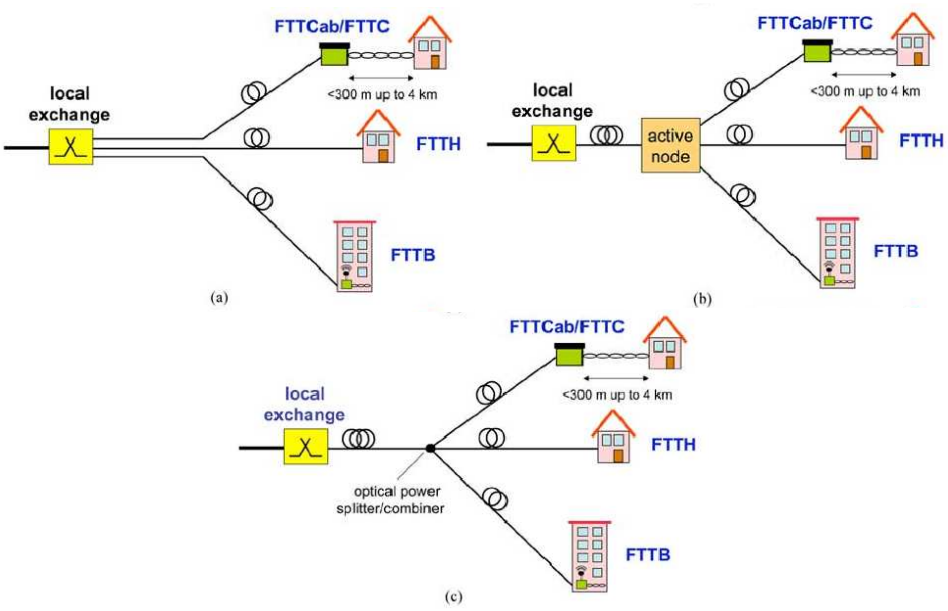


Figure 1.5: Fiber access network architectures.

fiber feeder link, the completely passive nature of the outside plant avoids the costs of powering and maintaining active equipment in the field.

This architecture has become very popular for the introduction of optical fiber into access networks, and is widely known as the passive optical network (PON). Typically, PONs are time-division multiplexing (TDM) single-channel systems, where the fiber infrastructure carries a single upstream wavelength channel (from subscribers to a central office) and a single downstream wavelength channel (from a central office to subscribers). IEEE 802.3av Ethernet PON (EPON) with a symmetric line rate of 10 *Gbps*, and ITU-T G.987 10 Gigabit PON (10G-PON) with an upstream line rate of 2.5 *Gbps* and a downstream line rate of 10 *Gbps* represent current state-of-the-art commercially available and widely deployed TDM PON access networks.

Wireless technologies

In the last fifteen years the progress in wireless digital data transmission has been huge and many different technologies have been proposed to provide access to the Internet. All these technologies can be divided in three main categories depending on the coverage distance provided (see Fig. 1.6).

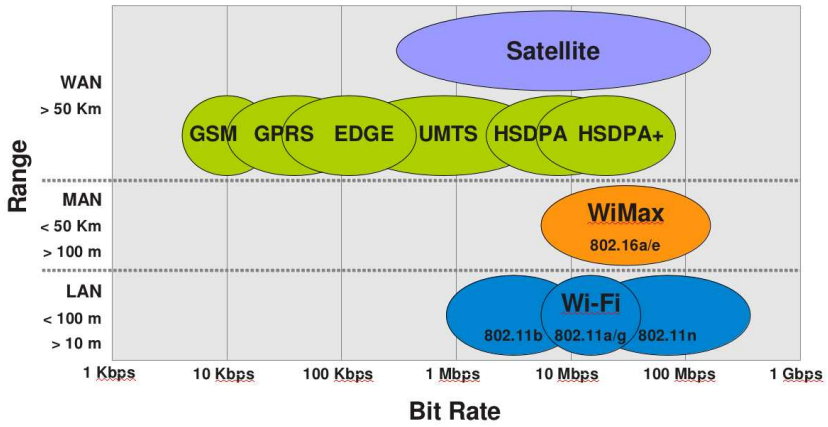


Figure 1.6: Wireless technologies overview: coverage vs speed.

Low-range access networks. Among the available wireless technologies, Wi-Fi represents the most famous and widely deployed low-range access technology to the Internet. The first 802.11 standard [22], approved by the IEEE in the early '90, defined physical and data-link layers for wireless LAN (WLAN) using both radio waves at 2.4 GHz, obtaining little success because of the limited achievable speed (up to 2 $Mbps$). In 1997 its evolution, standardized as IEEE 802.11a within the 5 GHz band and 802.11b within the 2.4 GHz, achieved data rates up to 11$Mbps$. More recently the IEEE 802.11g (2.4 GHz) and 802.11n (2.4/5 GHz) standards respectively support data rates up to 54 and 150 $Mbps$. Moreover, the latter can use two adjacent channels simultaneously (*channel bonding*), to double the maximum achievable rate.

The 802.11 MAC uses a multiple access technique called CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), which requires the stations to *sense* the channel before transmissions in order to detect if other stations are currently transmitting. This technique is similar to the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) used by the IEEE 802.3 at the base of the Ethernet standard. However, while with the 802.3 the stations are surely able to detect the presence of other transmissions on the shared channel, in wireless environments such detection is not always possible. Therefore, the CSMA/CA differs from the CSMA/CD in that the stations have to wait for a random time before transmitting also when the channel seems to be clear. Another important characteristic of the 802.11 MAC is the presence of a packet acknowledgement mechanism to ensure transmission reliability. Basically, when successfully receiving a packet, a station has to send a special packet (called acknowledgement or briefly ack)

to the transmitting station. In case the ack is not received, the packet is scheduled for retransmission.

The mechanisms described above have a clear impact on the QoS parameters observed in WLANs. For instance, the acknowledgement mechanism can cause large and variable transmission times in networks with several stations, which introduces more unpredictability in the end-to-end path [23]. Moreover, the 2.4 GHz band used by this technology is usually affected by interferences due to other devices or technologies working at the same frequencies (e.g. cordless phones, baby monitors, bluetooth devices, ZigBee devices, car alarms, microwave ovens, video senders, ...), which introduce unpredictable performance degradation.

Mid-range access networks. Currently, the most common wireless access technology considered for metropolitan area networks is WiMAX (Worldwide Interoperability for Microwave Access), which provides a signal radius of about 50 km. IEEE standard 802.16 defines the air interface and medium access control (MAC) protocol for a wireless metropolitan area network (WMAN), intended for providing high-bandwidth wireless voice and data for residential and enterprise use. The MAC protocol defines both frequency division duplex (FDD) and time division duplex (TDD). Downlink (DL) transmissions from a Base Station (BS) to Subscriber Stations (SSs) are conducted by point-to-multipoint broadband wireless access using a frequency channel for FDD or a time frame for TDD. Multiple SSs share one slotted uplink (UL) channel via TDD on a demand basis for voice, data, and multimedia traffic, and the BS handles bandwidth allocation by assigning uplink intervals based on requests from SSs.

The standard, first released in 2001, provided wireless transmission in the $10-66\ GHz$ band, with only a line-of-sight (LOS) capability. In 2003 the IEEE 802.16a standard added support for $2-11\ GHz$ band with non-line-of-site (NLOS) capability. The 802.16m version of the standard has been released in 2011 and supports data rates of 100 $Mbps$ for mobile stations and 1 $Gbps$ for fixed ones [24].

Even though WiMax, which implements also handover mechanisms, represents a good candidate to replace other mobile access technologies, it is mostly deployed as an alternative to cable and DSL to provide last-mile broadband access (see Fig. 1.7). Its advantage, indeed, consists in reducing deployment costs and in providing comparable data rates, considering the necessity to split the total bandwidth among multiple users.

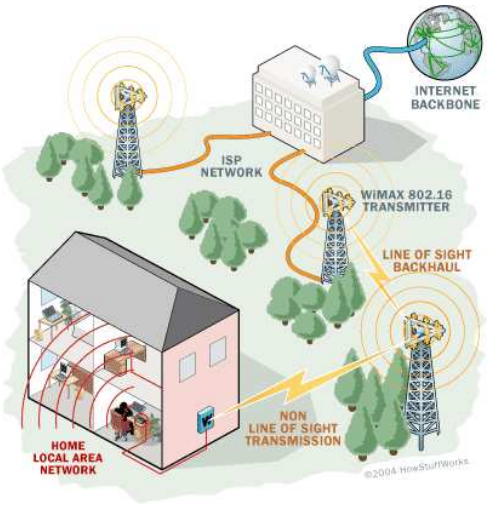


Figure 1.7: Wimax deployment scenario.

Wide-range access networks. As for wireless wide-range access technologies to the Internet, cellular and satellite networks represent most of the choices available today.

Cellular access networks. The cellular network is more and more used for Internet access since the introduction of the General Packet Radio Service (GPRS). The GPRS is actually a set of evolved services based on the GSM infrastructure, which allows packet-switching based communications on the circuit-switching network of the GSM. The spread of GPRS was mainly due to the fact that few modifications were required for the telecom operators to deploy this technology. In particular, it was necessary to add a node called Serving GPRS Support Node (SGSN) and a gateway called Gateway GPRS Support Node (GGSN). Moreover, these two components were also useful to progressively upgrade the network to the Universal Mobile Telecommunications System (UMTS) standard, which has now become very common in Europe for broadband Internet access. Over the last decade, the data rates achievable through the cellular network have increased from 80 *Kbps* for GPRS, to 236 *Kbps* for Enhanced Data rates for GSM Evolution (EDGE), then to 384 *Kbps* with basic UMTS, from 7.2 *Mbps* to 21 *Mbps* with the High Speed Downlink Packet Access (HSDPA) [25], and up to 81 *Mbps* with Evolved High Speed Packet Access (HSPA+). However, the complexity of the infrastructure of the cellular networks still results in unpredictable behavior of network QoS parameters.

Satellite access networks. A communications satellite is basically a microwave repeater station revolving around the earth in a specified orbit. Transmission of signals is performed via transponders working at frequencies larger than 1 GHz, which are usually capable of providing bandwidths up to 155 $Mbps$ [26]. Most satellites have multiple transponders (even over 70) which can be used simultaneously to gain higher bandwidth on demand and, since most of them act as simple retransmission stations, they cannot regenerate any previously sent signal. The satellite communicates with a ground station (hub), which consists of an outdoor unit (ODU) - usually a dish transceiver of $4-10$ m in diameter - and an indoor unit (IDU) responsible for demodulating, demultiplexing and reconstructing the incoming signal so that the tunneled IP datagrams can be sent to the high speed backbone Internet. Each satellite subscriber has a small ($65-240$ cm in diameter) low-cost dish antenna, which must have a direct Line-of-Sight (LOS) to the satellite, and a satellite modem which serves as an interface between the ODU and customer equipment.

The satellites can be categorized as GEO, MEO, LEO according to their orbit. Geosynchronous Earth Orbit (GEO) satellites are positioned at an altitude of 35786 km, to appear stationary with respect to certain point on earth, and cover nearly 33% of earth surface. Their main limitation is represented by the signal propagation delay, which is $250-300$ ms one-way. Medium Earth Orbit (MEO) satellites are positioned at $8000-20000$ km and have one-way delay of approximately $50-150$ ms. Low Earth Orbit (LEO) satellites operate at $350-2000$ km altitudes and with an acceptable $10-30$ ms one-way delay and take about 90 minutes for one complete revolution. Basically higher orbits imply higher round-trip-delay, launching cost, coverage, bit-error-rate (BER), signal attenuation and need for transmission power. Moreover, non stationary systems may require intersatellite links and switching protocols, making them much more complex and requiring a large amount of satellites to reach global coverage.

The first satellite Internet connectivity had a one-way hybrid model in which the uplink data stream was usually carried on a preexisting PSTN, ISDN or GPRS. Today, the bidirectional 2-way model is becoming more and more popular and offers a pure satellite connection to the Internet, where downlink uses broadcasting and uplink is multiplexed using e.g. TDMA.

Satellites use different frequencies for uplink (from ground station to satellite) and downlink (from satellite to ground station). These frequency bands are named S, L, X,

C, Ku, Ka and V-band which are controlled by IRFB (International Radio Frequencies Board) and the FCC (Federal Communications Commission). The C-band (3,7-4,2 GHz downlink, 5,925-6,425 GHz uplink) is the most common together with Ku-band (11,7-12,2 GHz downlink, 14,0-14,5 GHz uplink), but these bands are quickly becoming congested. Lately, new satellites have been tuned to use the Ka-band (17,7-21,2 GHz downlink, 27,5-31,0 GHz uplink) which currently offers almost 10 times as much capacity as the Ku-band [27], but suffers from rain attenuation and higher BER. The V-band, which is designed to be in range 40-75 GHz, is reserved for future use [28].

Final remarks

As shown above, the scenario of access technologies is highly heterogeneous and many factors may impact network performance. Hence, the evaluation of network performance at IP and higher layers represents a complex problem, which cannot be addresses without taking into account the peculiarities of the underlying technologies. It is also important to consider that, apart from the adopted technologies, QoS policies adopted by ISPs or implemented by user devices (e.g. broadband modems/routers) may heavily impact measurement results, thus making the performance evaluation task even more difficult.

1.1.2 Complexity of Internet applications

Today a change of paradigm is happening in the world of telecommunications: in the highly heterogeneous and dynamic context of the Internet, the user is becoming the real fulcrum. We are assisting to a radical change from the *Network-Centric* view to the *User-Centric* view. The user increasingly takes an active role in the network, promoting *peer-to-peer* (P2P) and *many-to-many* interactions. The variety of devices, together with his mobility, makes today the user a real network "*micro-operator*", sharing his broadband connection and providing both contents and network functionalities. We are therefore assisting to a shift toward the so-called *User-Centric Internet (UCI)*.

The transition to the UCI view is fostering the development of *multi-channel applications*. Such applications provide a single interface to perform heterogeneous activities, usually exploiting many communication channels. Since traditional approaches independently look at channels, the study, monitoring, and control of network traffic is becoming increasingly uneffective [29]. The main causes are the following:

- working with multi-channel applications we have also to cope with the problems of recognizing traffic flows associated with the same application and associating them with specific activities (e.g. signaling, video streaming, voice, file transfer, ...);
- transport layer port numbers are often randomly chosen or reused for non standard protocols;
- there is a trend toward an extensive use of encryption, obfuscation and encapsulation in communication channels. Therefore, it is necessary to find new techniques and analysis methodologies purposely designed for the properties of new generation applications.

Understanding and characterizing applications behavior and requirements in terms of network resources is fundamental in many networking fields:

- when operating *network planning and dimensioning*, the knowledge of traffic patterns allows to take better decisions by taking into account the performance perceived by the users for real applications;
- operators may implement *service differentiation* techniques able to appropriately treat each communication channel depending on its typology;
- *content delivery networks* can take advantage of a detailed knowledge of specific application to provide an efficient support for its data streams;
- once having a detailed knowledge of legitimate traffic, *intrusion and anomaly detection* techniques can more easily identify unwanted patterns.

In the context of multi-channel applications, being able to properly identify each channel allows to operate decisions at a finer granularity. For instance, an ISP providing both Internet access and telephony services could be interested in blocking or shaping only VoIP (Voice over IP) traffic pertaining to a specific competitor. With respect to a multi-channel application supporting also voice calls (e.g. Skype), the ISP may be forced to block/shape all its traffic. Such decision could force many users to change provider, thus resulting in monetary loss. Whereas, the ability to discriminate application activities can be used to selectively apply rules to them.

Taking into account the depicted scenario, the evaluation of network performance has to consider the characteristics of real applications in order to properly capture the behaviors observed by the users. Thus, by investigating the characteristics of new generation applications, specifically designed methodologies and techniques can be adopted.

1.2 Thesis contribution

In Sec. 1.1 we highlighted how the Internet scenario is evolving rapidly on both access technologies and applications complexity. Evaluating network performance in such conditions becomes challenging, because the number of possible interferences increases together with the gap between basic performance metrics and user quality of experience.

In the following sections we introduce the contributions of this thesis. We start by describing the contributions in the area of network traffic analysis and characterization. Thanks to these activities we have acquired the knowledge necessary to devise a new methodology to characterize multi-channel applications traffic. This, together with the analysis of existing access technologies, allowed us to better investigate the evaluation of network performance, in order to identify the most relevant low-level metrics and to evaluate related methodologies, techniques and tools. Exploiting the knowledge previously acquired, we addressed the evaluation of broadband access networks performance by conducting an extensive analysis of existing methodologies and techniques, which brought us to design an architecture to conduct large-scale measurements on real broadband networks. By implementing such architecture using two different approaches we devised new methodologies and techniques to enable its deployment.

1.2.1 Characterization of new generation applications

To better address the study of new generation applications, we proposed the definition of a novel methodology for the characterization of multi-channel applications working at different abstraction layers. The methodology is based on a multi-layer traffic inspection and a decomposition approach counting four layers:

- ***host***: aggregates the whole traffic pertaining to a single host;
- ***service***: groups together packets having the same transport protocol and IP address-port pair;

- ***biflow*** (bidirectional flow)[2]: aggregates packets having the same 5-tuple, where source and destination can be swapped, and represents one application channel;
- ***packet***: looks at the properties of each packet (e.g. size, inter-packet time, payload, ...).

According to this decomposition, a biflow corresponds to a channel and, aggregating traffic at each layer, data is typically inspected at lower layers (e.g. packet-sizes distribution at host layer).

Combining information collected at these layers can reveal useful patterns in host interactions, traffic flows statistics, congestion prevention/reaction mechanisms, overlay communications topologies, geolocalization aspects, etc. For instance, if a host is running eMule on TCP port 80 and UDP port 53 with obfuscation enabled, it would be difficult to identify it by independently looking at biflows exploiting port numbers, payload content or flow statistics. Whereas, characterizing the correlation between host and biflow layers could reveal patterns peculiar to the application (e.g. TCP/UDP biflows ratio, connections temporal sequences, ...). Therefore, correlating multiple channels allows to better identify the application generating them and to identify the application level activities conducted on each of them.

The proposed methodology identifies different activities performed by multi-channel applications, thus helping in their characterization. After having characterized the main properties of traffic generated by specific activities (e.g. in terms of packet size and inter arrival distributions), it is then possible to evaluate their performance in different scenarios by conducting active measurements reproducing similar traffic patterns.

1.2.2 IP networks performance evaluation

After acquiring the necessary knowledge about multi-channel applications and their traffic patterns, we performed an extensive analysis of metrics for network performance evaluation, in order to identify the most relevant parameters to measure for benchmarking access networks[3].

The *communication network metrology* science identifies the following parameters as exhaustive for the characterization of performance for a packet-switching network: one-

[2]Source and destination roles are related to the first packet.

[3]Most multi-channel applications are utilized by users when they are at home.

way delay, delay variation (or jitter), round-trip time (RTT), packet loss, packet reordering, route and bandwidth [30]. Such parameters mostly refer to one direction of the communication between two end systems and can be measured using both active techniques, which purposely generate synthetic traffic, and passive techniques, which opportunistically take advantage of user-generated traffic. The IPPM (IP Performance Metrics) group of IETF (Internet Engineering Task Force) proposed a standard definition for part of them, while is still working on the others. In the context of Internet broadband access networks, the DSL Forum, supported by the Verizon ISP, has proposed a standardization of infrastructures and techniques to monitor their performance [16, 31]. Furthermore, many methodologies and techniques are available to measure part of the parameters cited above with different levels of accuracy and intrusiveness [32, 33].

In general ISPs advertise their broadband plans only in terms of download and (sometimes) upload speeds. Such metric is not sufficient to describe the performance that user can expect when using various applications. When subscribing to a connectivity plan, the user should be able to decide which offer better responds to his needs. Driven by this requirement and by experimental results conducted on real scenarios, we contributed to the selection of a set of parameters to be presented to the user in a standard format. Such format, in analogy with "Nutrition Labels" for food items, reports both low-level parameters - represented as ranges of possible values - and high-level concepts (e.g. time to download a song, video streaming quality in terms of frames rate) [34]. By analyzing the parameters proposed in the label on several broadband connections in the USA, we demonstrated that often Internet plans having the same advertised characteristics present big differences if considering all the proposed metrics.

Once conducted the previous studies we acquired the knowledge necessary to address the evaluation of performance in the specific context of broadband access networks.

1.2.3 Evaluation of broadband access networks performance

Focusing our attention on broadband access networks, we performed an extensive analysis of existing projects addressing their performance evaluation. We detected that none of them consider all the performance metrics we selected and each of them adopts different techniques and methodologies to measure the same parameters [35]. As a result of the analysis we defined a taxonomy dividing all the approaches in three macro-categories depending on the device hosting the measurement tools:

- **server-based**, in which the measurements are conducted from a server in the core of the network, can be further divided in three categories:
  - *service-based*, in which passive measurements are performed from servers directly offering a specific service (e.g. Youtube [36]);
  - *core-based* in which active measurements are conducted from dedicated servers [7, 37];
  - *isp-based*, which provides passive measurements conducted by the ISP itself [38, 39, 9].
- **host-based**, in which passive or active measurements are performed from users' devices (e.g. pc, mobile phone, tablet, ...), can be further divided in two categories:
  - *client-based*, which requires the installation of a software client on users' pc [40, 41, 42, 43, 44, 45];
  - *web-based*, in which the execution of a software component inside the Web browser is required, can be further divided in two sub-categories:
    - *plugin-based*, based on a browser plugin [46];
    - *embedded*, which embeds the measurement tool into a *Flash* [47] or *Java* [8, 4, 48] container.
- **router-based**, in which both active and passive measurements are conducted directly from the router acting as gateway to the Internet for the local network [49].

Server-based approaches resulted to be the most limited. When adopting passive measurements, since many applications limit their own traffic, they do not guarantee the saturation of the link. When using active techniques, due to dynamic IP addressing, they are unable to to determine whether repeated measurements of the same IP address are in fact measuring the same access link over time.

Both host-based and router-based approaches, operating from the users' point of view, are considered the most accurate and scalable. Among them we identify as the most promising the following approaches: client-based, plugin-based, and router-based. Their advantage consists in the possibility to execute many repeated measurements over time on the same access link, which guarantees more statistically valid results. All these three

approaches have some peculiar advantages and drawbacks. The first two approaches can easily reach many installations, allowing to obtaining results at a finer geographic granularity. On the other hand, their measurements are affected by many uncontrollable confounding factors (e.g. concurrent processes, cross-traffic, ...) and can be executed only when the computer hosting the tool is switched on. The third approach, being able to observe all the cross-traffic and operating in a completely controlled environment (i.e. the router), is able to overcome most confounding factors and to obtain more accurate measurements. Being usually always active, it also allows to execute continuous and periodic experiments over long time periods. On the other side, requiring the installation of a customized device into the local network, it results to be less scalable and more indicate to collect results at a higher geographic granularity.

As a result of the preceding analysis we identified the ideal characteristics for an architecture able to evaluate the performance of broadband access networks on a large scale. Following such guidelines, we designed and implemented two architectures respectively adopting the router- and client-based approaches, in order to evaluate their potentialities and the eventual advantages deriving from a combined approach. Thanks to this activity we devised new methodologies and techniques to cope with several challenges arose during the different phases from the design to the deployment. Thanks to the data collected through these architectures, we also found some interesting results about real access networks, by identifying and characterizing relevant issues affecting most of them.

1.3 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2 we report the research activities we performed on both the analysis and characterization traffic generated by new generation applications and the identification of relevant metrics, methodologies, techniques and tools to evaluate network performance.

Our research activities on the evaluation of broadband access networks performance are reported in Chapter 3, where we propose a taxonomy of existing approaches and define the guidelines to build an architecture with ideal characteristics to measure access networks performance on a large scale.

In Chapters 4 and 5 we respectively describe the client- and router-based architectures we designed, implemented and deployed, by detailing the challenges we faced and the

solutions we adopted to solve them. For both we also present some findings obtained by analyzing the data collected by their deployments.

Finally, we draw our conclusions in Chapter 6.

Chapter 2

Characterizing applications to evaluate IP networks performance

Since understanding how new generation applications behave is important to evaluate their performance, in this chapter we face their characterization by analyzing the literature and by proposing a novel methodology. This allowed us to better investigate the evaluation of network performance, in order to identify the most relevant low-level metrics and to evaluate related methodologies, techniques and tools.

2.1 Characterization of new generation applications

In the last decade we assisted to a radical change of Internet paradigms: the user, which has always been a simple content consumer, has become the real fulcrum of the infrastructure [50, 51]. It opened, therefore, a completely new scenario in terms of services and applications, in which the user is involved in new forms of communication combing multiple heterogeneous channels (voice, video, chat, virtual worlds, ...). We proposed for such applications the following definition:

Definition An application is considered *multi-channel* if it provides a single interface to perform heterogeneous activities exploiting multiple communication channels.

2.1.1 Related work

We group existing multi-channel applications into four categories:

- *Communication platforms*, which allow multimedia interactions among users (e.g. Skype, Microsoft Live Messenger);

- *Social networking*, which provide highly interactive multimedia services (e.g. YouTube, Facebook, Flickr);
- *Virtual worlds*, in which users establish social interactions in a virtual reality (e.g. Second Life) or play a massively multi-player online game (e.g. World of Warcraft);
- *Cloud computing*, which provides remote access to complex applications (e.g. Google Docs, EyeOS).

In the following paragraphs we briefly detail some relevant works found in literature about the characterization of application belonging to such categories.

Communication platforms. One of the first communication platforms that has attracted the attention of the research community is Skype [52]. Skype is an integrated communication platform which makes use of proprietary protocols and cryptography. For this reason, this platform presents several aspects of interest which were deeply analyzed in several studies. Its main features were investigated in [53] with a methodology which took into account at the same time the generated traffic and the system calls invoked by the application. The experiments were performed in a controlled environment (i.e the application was launched on several monitored machines) focusing on the several phases carried out by Skype like login, firewall traversing, call and conference setup and file transferring. On the contrary, authors in [54] reported information about the topological properties of the Skype overlay network mainly focusing on its super nodes (i.e nodes which act as relay for the others). Again, the experiments were conducted in a controlled environment. Another interesting aspect is to identify the traffic generated by Skype. Such topic was first investigated in [55], where the authors made use of two distinct techniques. The first technique was based on recurrent patterns inside packets payload. The second one took into account the statistical properties of the generated traffic in terms of dimension and arrival rate. Both techniques were validated with real traffic collected in two sites: inside a campus network and an Internet Service Provider. The identification problem was the topic of another work too. Authors of [56] detailed a methodology to identify both the traffic generated by Skype and GoogleTalk[57], comparing the results with those obtained from a model well known in literature that was developed for HTTP traffic.

In [58], the authors proposed a methodology to register the evidence of relay operations (i.e traffic forward for users not able to directly communicate due to firewalls/NATs) inside the Skype traffic generated by some users. The technique was validated in a controlled environment with real traffic.

Authors in [59] deepened the congestion control mechanism adopted by Skype. In this work, several adverse circumstances were emulated to register the Skype reaction. In order to emulate the network behavior perceived by the application, authors extracted and analyzed information obtained from the graphical layout like round trip time, jitter, video resolution and frame rate.

The works [60, 61] proposed characterizations of the Skype traffic. The authors performed separately the characterization of the signaling and video traffic with real traces. Furthermore, they studied the Skype reaction in terms of generated traffic to different network conditions making use of a laboratory testbed.

Other analysis of the signaling traffic are reported in [62]. In this work, authors reported results of passive measurements. They showed how signaling operations cause a reduced amount of traffic in absolute terms, which however represents an important percentage of the whole traffic generated by the application.

In [63], authors compared Skype and GoogleTalk in terms of quality of the audio perceived by the user when the network conditions change. They focused on similarities and difference among codecs and internal mechanisms used by the two applications to avoid the audio quality degradation when the network conditions are adverse. Furthermore, they presented a prototype which made use of the positive aspects of both applications.

Regarding the audio quality experienced by the users, [64] proposed a model based on parameters extracted from network traffic able to quantify the users satisfaction.

Social networking. Regarding the social networking sites which provide multi-channel interactions, one that has received attention from the research community due to its popularity is Youtube [65]. It is a platform for content sharing which allows the users to upload and to explore audio and video files. At the same time, users received additional information on the viewed contents and related contents. In this way, the users have the possibility to interact with the system thanks to different channels.

Several aspects related to this platform were studied in literature. In [66] it is proposed the characterization of traffic exchanged by Youtube and users of a campus, focusing

on temporal evolution of number of users, video demand, download speed, amount of downloaded data. Furthermore, they showed in what this type of traffic differs from traditional Web traffic. Monitoring over several months the information reported on the Youtube site, the authors inferred additional information about videos like coding bitrate, duration, dimension, ranking, etc.

Regarding the analysis of the information reported on the Youtube site, another interesting work is [67]. In this study, the authors collected information related to available contents from not only the Youtube portal but even Daun UCC, the most popular video sharing system in Korea. The comparison of the results obtained from the two platforms allowed to generalize part of the detailed findings.

On the other side, authors in [68] reported an analysis of the exchanged traffic between a campus network users and Youtube focusing on parameters like video popularity, the request rate per video and per user and on several temporal scales, session duration, etc.

Regarding other interesting studies about social networking sites, it is worth to notice [69] and [70]. The first one focused on Facebook[71], a social networking platform which has attracted a relevant amount of users. The authors developed a Facebook application by taking advantage of the Facebook's API. Thanks to this application, they collected information related to 8 million users. The detailed results shed light on the existing dynamics between the Facebook community and the third party developed applications in terms of user reaction time and applications peculiarities related to online games.

In [70], authors analyzed the information contained on 4 social networking sites: Flickr [72], YouTube [65], LiveJournal [73] and Orkut [74]. The analysis is focused on main aspects of these social networks, shedding light on dynamics already observed in other users communities.

Virtual worlds. Among the multi-channel applications allowing users to interact in a virtual environment, it is worth to notice Second Life, due to the high number of users. In [75], the authors performed analyses to understand the effect on the network of particular actions engaged by the avatars (i.e. the virtual characters controlled by the users) in the virtual world. Network parameters like throughput, packet size, inter-arrival time and the generated traffic volume were related to particular actions in the virtual world like a tour inside crowded or isolated places, music listening, and the usage different transport systems. This work took into account even requirements and utilization of the bandwidth.

In [76], the authors investigated spatial and temporal dynamics in SecondLife. In order to obtain this information, the authors developed a crawler tool which simulates an avatar. This avatar tours continuously the virtual world of SecondLife allowing the collection of information like number and name of the other avatars, objects, traveling duration, etc.

A similar methodology was used in [77]. This paper shows results of data collected during a one month of a crawler's activity. This study shows how some characteristics of the virtual world like people density per zone, social relations, etc. are really similar to the ones of the real world. Authors suggested some improvements too.

Another set of successful multi-channel applications in recent years is the IP television especially the ones based on peer-to-peer system. Applications like TVAnts [78], PPLive [79], SopCast [80] e PPStream [81] become day by day more popular since they allow users connected to the peer-to-peer network to exchange multimedia content in streaming mode. Sometimes, these mechanisms overcome limitation imposed by countries legislation and copyrights.

In [82], the authors reported several characteristics of the generated signaling and multimedia traffic, the upstream and downstream traffic, etc. Furthermore, information about the behavior of several peers in the network is extracted, shedding light on aspects like the number of peers involved in active data exchanging, peer's permanence, etc.

In [83], the authors detailed an architecture for multimedia peer-to-peer streaming and its main features like implementation's simplicity and reachable performance. Preliminary results were obtained with a prototype.

[84] proposed a methodology to infer parameters like peer connection duration and re-connection continuity, content production starting from the clients exchanged information. Thanks to this methodology, authors collected information about the whole peer-to-peer network.

Cloud computing. Finally, cloud computing was the focus of several papers. It is worth to cite [85] and [86]. In [85], the authors claimed that not always cloud computing services need an intensive communication among pool of servers and, thus, they could be implemented by using small data centers geographically distributed. This choice would reduce costs while improving scalability and performance. On the other side, in [86], the authors detailed an open source architecture for cloud computing services called Eucalip-

tus. This architecture allows the users to execute and control virtual machines on several different real machines. The paper analyzed the main aspects of the architecture and trade-offs among modularity, portability and simplicity.

Final remarks. The analysis of the scientific literature on multi-channel applications shows how the approaches adopted are too specialized for each application group taken into account. No methodology has been proposed to study such applications independently from the particular scenario. Thus, no general conclusions have been drawn and no common properties have been detected about these applications.

Therefore, it is necessary to find new techniques and analysis methodologies purposely designed for the properties of emerging applications. For example, considering the relations between channels belonging to the same application can reveal behavioral patterns otherwise not visible: our approach starts from this assumption.

2.1.2 A new methodology for characterizing multi-channel applications

We propose the definition of a novel methodology for the characterization of multi-channel applications working at different abstraction layers. The methodology is based on a multi-layer traffic inspection and a decomposition approach, as depicted in Fig. 2.1, counting four layers: *host*, *service*, *biflow* (bidirectional flow) and *packet*. The *host* layer aggregates the whole traffic pertaining to a single host. The *service* layer groups together packets having the same transport protocol and IP address-port pair. The *biflow*[1] layer aggregates packets belonging to the same channel (i.e. having the same 5-tuple, where source and destination can be swapped). Finally, the *packet* layer looks at the properties of each packet (e.g. size, inter-packet time, payload, ...). According to this decomposition, a biflow corresponds to a channel and, aggregating traffic at each layer, data is typically inspected at lower layers (e.g. packet-sizes distribution at host layer).

Combining information collected at these layers can reveal useful patterns in host interactions, traffic flows statistics, congestion prevention/reaction mechanisms, overlay communications topologies, geolocation aspects, etc. For instance, if the host in Fig. 2.1 is running eMule on TCP port 80 and UDP port 53 with obfuscation enabled, it would be difficult to identify it by independently looking at biflows exploiting port numbers,

[1]Source and destination roles are related to the first packet.

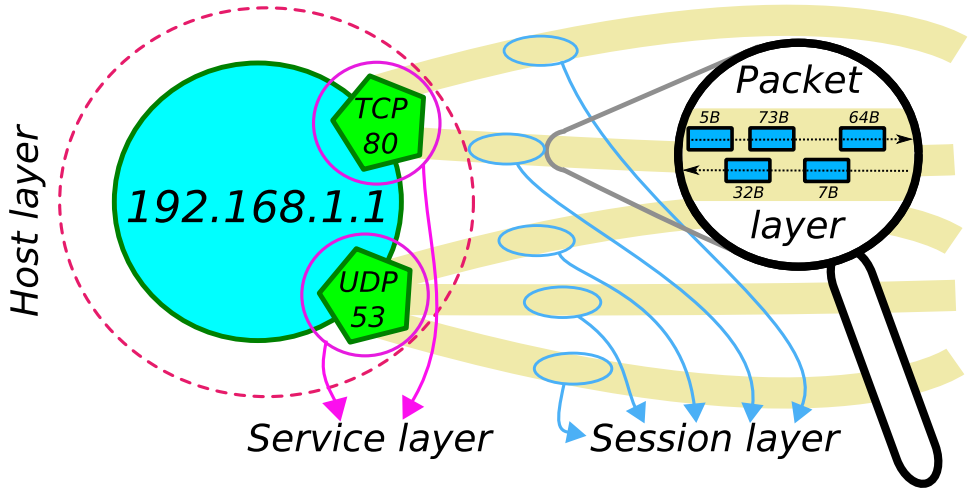


Figure 2.1: Analyzing traffic at different layers.

payload content or flow statistics. Whereas, characterizing the correlation between host and biflow layers could reveal patterns peculiar to the application (e.g. TCP/UDP biflows ratio, connections temporal sequences, ...). Therefore, correlating multiple channels has two main benefits:

- by looking at many biflows belonging to the same application it is possible to detect the application itself;
- being aware of an application running on a particular host/service can help in associating a new flow to it, and to identify the related activity.

2.1.3 Experimental analysis: a proof of concept

We applied the proposed methodology to Skype in order to prove its feasibility and benefits. Skype represents an interesting case study since it works on a super-peer based P2P overlay architecture, its communications are mostly encrypted and the adopted protocols are secret.

We used TIE [87] to gain knowledge of the traffic associated to each channel (see Tab. 2.1), and we discovered several patterns[2] at different layers. We found that, differently from traditional applications, Skype listens for both TCP and UDP connections on the

[2]Since Skype exposes different patterns depending on network configuration, we present a preliminary analysis of the generic super-peer case: public IP address and no firewall restrictions.

Table 2.1: Skype traffic at biflow layer.

Activity	**proto**	**src port**	**dst port**	**up pkts**	**down pkts**	**up bytes**	**down bytes**
Super-peer signaling	udp	33837	26137	2	2	71	29
	tcp	51236	26137	161	97	19 k	9 k
Normal p2p signaling	udp	57046	33837	1	1	31	123
		33837	11229	3	3	527	497
		33837	17983	1	4	22	5 k
File transfer	udp	13524	33837	243	247	6 k	123 k
Call	udp	33837	13524	3 k	4 k	493 k	484 k

same fixed port number (33837), randomly chosen at installation time[3]. Moreover, when connected to the P2P network, it always has at least one persistent TCP connection with a super-peer.

By analyzing ***host***-layer information, we found that port numbers are used more than once on a short period: this easily reveals which services the host is listening for (i.e. port 33837). Then, considering the ***service*** layer, we discover that the application uses port 33837 also for outgoing UDP connections. This also reveals both the UDP and TCP listening ports of peers on the other side. At ***biflow*** layer we see that signaling traffic is mostly composed by many short UDP biflows revealing a few different patterns:

- some of them consist of only two packets (one per direction) of predictable size;
- others present a single query packet and some response packets;
- others present few packets in equal number for upstream and downstream directions with similar cumulative sizes.

On the other side, file transfers present almost the same number of packets in both directions, but most bytes fall only in one of them. Finally, voice calls reveal a symmetric pattern in transferred data. At the ***packet*** layer, as shown in Fig. 2.2, the distribution of file transfer and voice-call payload sizes are significantly different. Combining the previous observations allows to identify Skype and its activities. For instance, the detection of many short UDP biflows related to the same service that show known patterns at packet level, allows to easily infer the Skype random port number. After that, it is straightforward to

[3]It also listens on ports 443 and 80 to provide connectivity in presence of firewalls.

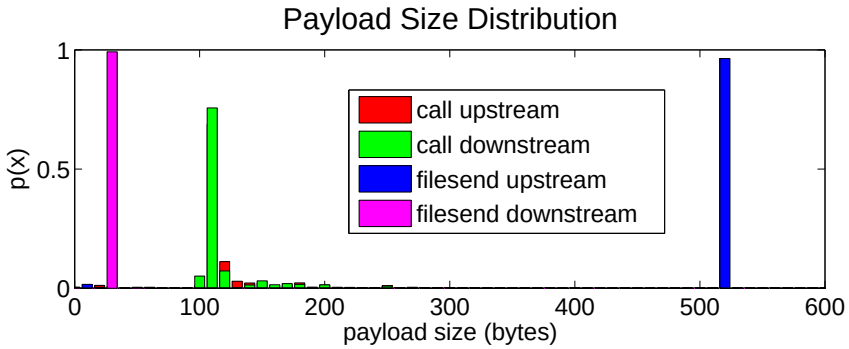


Figure 2.2: Skype: voice call vs file transfer.

label all its communications in a port-based fashion (also for incoming TCP connections). Moreover, by inspecting the packet-size distribution of each biflow, it is also possible to discriminate between file transfers and voice calls.

As a final consideration, we can state that the proposed layered methodology can be used to detect the presence and to identify behavioral patterns of a Skype client running on a host. Moreover, combining information taken at different layers allows us to identify the specific activities performed by the application and to isolate the associated communication channels.

2.2 Evaluation of network performance

The methodology proposed in Sec. 2.1 can be used to identify different activities performed by multi-channel applications, thus helping to characterize them with higher accuracy. It applies to any multi-channel application and can detect behaviors and aspects otherwise not visible. Once characterized the main low-level properties of traffic generated by specific application-level activities, it is possible to identify the most relevant metrics to evaluate their performance.

After acquiring the necessary knowledge about multi-channel applications and their traffic patterns, we performed an extensive analysis of metrics for network performance evaluation, in order to identify the most relevant parameters to measure for benchmarking access networks, since most new generation applications are heavily utilized in home networks.

2.2.1 IP performance evaluation metrics and techniques

Now that the Internet is used for applications with QoS constraints (real-time applications, etc.), ISPs and their customers negotiate QoS levels through a Service Level Agreement (SLA). In this context, metrology is useful in verifying that SLAs are met.

Network measurements can be performed actively, purposely generating synthetic traffic, and or passively, opportunistically taking advantage of user-generated traffic.

Passive techniques are carried out by observing network traffic flows. They consist of capturing packet headers and analyzing them. The best example of a capture tool is tcpdump, which is based on the libpcap library [88]. These techniques do not add network traffic, thus result to be network friendly. Passive measurement can be done on two levels:

- *Microscopic level*: in which measurements are performed on each packet traveling across the measurement point;
- *Macroscopic level*: in which measurements are performed on flows, where aggregation rules are necessary to match packets into flows.

Passive measurement techniques are particularly suitable for traffic engineering because they show flow dynamics and distribution. The main problem with passive measurement is the data volume. The volume of captured data can become very large on high-capacity links. Moreover, it is hard to obtain end-to-end measurements passively: the presence of traffic traveling between two measurement points cannot be ensured and matching measurements that are performed from different vantage points is difficult. Consequently, passive measurements are usually used to determine metrics characterizing one particular network element.

On the other hand, active measurements are performed by sending probe packets to the network. The measurement flow travels from source to destination. Upon reaching its destination, it is possible to calculate metrics by analyzing the packets. Active measurements can determine the end-to-end QoS experienced by a generated flow for a particular path and then estimate the QoS as it is seen by applications. Additionally, active measurements offer the flexibility to send probe packets streams with particular properties (bitrate, packet size, etc.). Their main drawback is that additional network traffic is introduced. This "intrusive" (or "invasive") characteristic can potentially modify the properties that are trying to be measured. First, it can result in measurement

errors or bias; second, it can lead to network overload. Measurement traffic thus has to be limited to avoid network disturbance and measurement errors.

Since active measurements can better measure network performance, in the following we focus our attention on them.

Active measurement techniques and tools

Active measurements consist of sending probe packets into the network from a source host (probe sender) to a destination host (probe receiver). By choosing particular properties at departure (packet size, inter-departure time, bitrate, etc.), it is possible to calculate metrics by analyzing the probe stream characteristics (arrival time, inter-arrival time, etc.) at the destination. Thus, one can determine end-to-end metrics (from the source to the destination).

Active measurement tools can be classified into the following categories:

- *Cooperative*: which consist of separate sender and receiver programs that are respectively installed on the source and destination hosts.
- *Non-cooperative*: which consist of only one program including both the sending and receiving tasks and avoids to install dedicated software on the target end-system.
- *Certain*: which are based on forcing routers and hosts to generate ICMP replies, whose applicability may be limited by firewall rules or router configurations.

Some tools require having a listening TCP socket on the target host (e.g. Web servers). They use TCP behavior to force the target host to send packets back (e.g. by sending SYN packets to the target). Since sending large streams of SYN packets can be interpreted as a denial of service attack, many network sites often use a firewall to filter or limit incoming SYN traffic, thus disrupting such approach.

To obtain accurate results, certain active measurement techniques require strong time-related constraints to be achieved:

- Timing accuracy in sending probes;
- Accurate time stamping of probes upon arrival;
- Accurate time synchronization of the source and the destination to allow clock comparison between the two hosts.

Thus, active measurement techniques will often induce strong constraints on clocks, and measurement errors will result from clock characteristics. A parameter is always measured at a specific protocol layer. If the parameter needs to be used at another layer, the result has to be modified in order to take into account the protocols operating at intermediate layers. We discuss this "encapsulation effect" in the following paragraph.

Encapsulation effect. Some measurement tools work at protocol layers lower than application layer. The user must be aware of the significant differences that can exist between the measured value and the real value of the parameter at the application layer. A measurement result could also appear incorrect when a parameter value is known at another layer. For example, the capacity of a channel is often known at the data link layer, while measurement tools usually work at IP layer, which makes difficult to analyze their results. To avoid such problem, it is recommended to measure a parameter at the user level.

Timing considerations. Active measurement techniques often result in constraints to the clocks in the measuring end-systems (clock accuracy, clock stability, etc.).

Clock uncertainty relates to the following parameters:

- *Synchronization error*: measures the extent to which two clocks agree on what time it is;
- *Accuracy*: measures the extent to which a given clock agrees with Coordinated Universal Time (UTC).
- *Resolution*: measures the precision of a given clock.
- *Skew*: measures the change of accuracy over time.
- *Drift*: measures the variation of skew over time.

A computer contains two clocks: a hardware clock and a system clock. The hardware clock allows the computer to keep track of time when the computer is turned off and is used when the computer is booted up to update the system clock. The latter is implemented as a counter of timer interrupts that are generated by the computer quartz oscillator. This value characterizes the granularity of the clock and is called the "tick", which is usually of 10 ms.

To improve such granularity the OS uses the clock cycle register (which counts CPU clock cycles) to interpolate between the interrupts. This technique requires knowing the number of CPU cycles per time unit. This kind of clock is prone to error:

- Changes in frequency of the quartz oscillator introduce skew.
- Being the measurement of CPU clock cycles per time unit based on the oscillator, the interpolation inherits its skew.
- The latter measurement suffers from integer arithmetic effects.

Clock synchronization is a complex problem and can be achieved by using Global Positioning System (GPS) cards, radio clock receivers, or Network Time Protocol (NTP) servers. The first approach is the most precise and expensive solution, requiring the installation of exterior antennas, while cheaper radio clock receivers are not accurate for many reasons. Using NTP servers is then the most common solution for clock synchronization, which is expected to be accurate to about 10ms on a WAN. NTP synchronization, while not perfect, can nevertheless offer significant accuracy given the scale on which measurements are made. For example, when measuring delays to the order of hundreds of milliseconds, a synchronization error of 10ms may be sufficient.

Metrics for evaluating network performance

The IPPM (IP Performance Metrics) group of IETF (Internet Engineering Task Force) is active in defining a standard definition of network parameters. We report in the following paragraph a complete description of such parameters.

One-way delay. The one-way delay is the time it takes a packet to go from source to destination. It includes propagation delays, transmission delays, and queuing delays in intermediate systems (routers, switches).

To measure one-way delay, a packet is stamped with the current time and sent to the destination. At the destination, the packet timestamp and the destination clock are read out. The delay is then equal to the difference between the two values. This technique implies that the clocks of the two end-systems are synchronized. One-way delay measurement is implemented in owping/owampd [89], QoSMet [90] and D-ITG [91]. These tools can measure one-way delay along the forward and reverse path.

One-way delay variation. Delay variation, often referred to as "jitter", is a key metric for many applications. For example, a high delay variation can disrupt the transfer of continuous media of voice.

Delay variation is computed as the difference between the delay of two consecutive packets, thus it requires the measurement of one-way delay, but it does not need clock synchronization because errors will cancel each other when the delay difference is calculated. One-way delay variation measurement is implemented in QoSMet [90] and D-ITG [91], which can measure this metric along the forward and reverse path.

Round-trip time (RTT). The RTT, often simply referred to as "latency", is the delay from the source to the destination and back.

To measure RTT, a packet is stamped with the current time and sent to the destination. When the packet is completely received at the destination, it sends a corresponding response packet back to the source. The RTT is equal to the difference between the receiving time at the source and the time stamp value. The measurement is easier than one-way delay measurement, because there is no issue with source and destination clock synchronization and clock skew is negligible due to the measurement time-scale. The best known RTT measurement tool is ping, which is a non-cooperative tool that measures RTT using ICMP echo-request probes. When a host receives such packets, it sends back echo reply packets that are the same size as echo packets. RTT corresponds then to the time elapsed by the ICMP reply packets to come back.

Packet loss. The reliability of a path is expressed by the packet loss rate. This metric is equal to the number of non-received packets divided by the total number of sent packets.

According to IETF recommendations, detection of non-received packets is performed using timeout values. If the packet fails to arrive within a reasonable period of time, the packet is considered lost. One-way packet loss measurement is implemented in owping/owampd [89], QoSMet [90] and D-ITG [91], which can measure loss along the forward and reverse path. Ping is the better known tool for measuring losses. It determines round-trip losses of ICMP probe packets and is not able to distinguish a difference between a loss that occurs in the forward or reverse path.

Packet reordering. Ordered delivery of packets is essential for real-time media streaming applications and the TCP protocol. Packet order must be considered in play-out buffer dimensioning of real-time streaming applications. Moreover, out-of-order packets can cause TCP senders to unnecessarily retransmit packets and/or the congestion window to increase at a slower rate than normal.

The reordering metric measurement is implemented in owping/owampd [89], QoSMet [90], which can measure it along the forward and reverse path by sending streams of packets and by analyzing packets sequence numbers.

Route. The route is an ordered sequence of nodes that represent a path from a source to a destination crossed by the exchanged traffic. Each node is identified by its IP address. A complete route from a source to a destination may consist of one or more IP addresses.

The route from source to destination can be determined by taking advantage of the "time to live" (TTL) field of IP packets. Routers have to decrease it by one unit when processing the packet. If the router decreases the TTL field to zero, it discards the packet. In this case, the router sends back an ICMP "time exceeded" message informing the sender that the packet was dropped. To trace the route to the destination, the source first sends UDP packets to the destination with TTL fixed to 1, which is then discarded by the first router on the path that sends back an ICMP message to the source to inform it of the drop. Next, the source sends packets with TTL fixed to 2, and so on. It proceeds in this way until it receives a reply from the destination.

The reordering metric measurement is implemented by the well known traceroute tool [92] and by many different variants of it.

Bandwidth. A network path from source to destination represents a sequence of store-and-forward links, and assuming that the path is fixed and unique (no routing changes or multipath forwarding occur during the measurements), different bandwidth measurements can be performed: per-hop capacity, end-to-end capacity, available bandwidth and bulk-transfer capacity.

Bandwidth is often known at the data link layer and corresponding capacity at the IP layer depends on the size of the IP packet relative to the layer 2 overhead.

Per-hop capacity. The per-hop capacity (also raw bandwidth) defines the maximum rate at which packets can be transmitted by a link.

There exist two main models to measure per-hop capacity:

- *one-packet model*: also known as the Variable Packet Size (VPS) model, it does not account for intra-flow queuing delay and assumes that delay is linear with respect to packet size, then proposing to express the packet delay as a function of the packet size and the capacity of each crossed link. The method consists of determining packet delays of different sizes from a source to each router that is on the path towards the destination and inferring capacities using linear regressions. This technique is implemented by Pathchar [93], Bing [94], clink [95], and pchar [96], which are non-cooperative tools. In all these implementations, the source reaches each router by sending probe packets and exploiting the TTL field of IP packets and ICMP "time exceeded" messages as traceroute. This technique allows the sender to measure the RTT to each router. This mechanism is repeated for different packet sizes.

- the *multi-packet model* was introduced to compensate the high cost of one-packet model in terms of the traffic that must be sent and it is focused on intra-flow queuing delays. Also this method generates a non-negligible amount of traffic, because it must send packets of many sizes and many packets per size. However, it does this once for the entire path, while the one-packet technique does this once for every link. This technique is implemented in Nettimer [97], which is a non-cooperative tool. In practice, the largest possible non-fragmented packet with a particular TTL field is sent, and it is immediately followed by the smallest possible packet. The smaller packet almost always has a lower transmission delay than the larger packet's transmission delay on the next link. Then the smaller packet queues continuously after the larger packet. The tailgated packet is dropped at a particular router due to its TTL. The tailgater can then continue without queuing to the destination.

End-to-end capacity. The end-to-end capacity of a path corresponds to the minimum per-hop capacity of the links composing it. The packet-pair dispersion method is able to measure it by generating two back-to-back packets (a packet-pair) which will be spread out in time when they arrive at the narrow link. The method accuracy is limited by many factors: cross traffic, packet size, time related errors and uncertainties when re-

ceiving a packet. This technique is implemented by both cooperative and non-cooperative tools: bprobe [98], sprobe [99], pathrate [100], and Nettimer [97].

Available bandwidth. The available bandwidth of a link, during a certain time interval, corresponds to the difference between the per-hop capacity and the utilization rate. As for the capacity, The available bandwidth of a path is represented by the minimum available bandwidth of all the links composing it.

The techniques to measure the available bandwidth can be classified into three categories:

- *Packet Train Dispersion* (PTD): consists of sending $N > 2$ back-to-back packets of size L (a packet train of length N) to the receiver. The rate at which the N packets are sent must be larger than the available bandwidth of the tight link. When arriving at the receiver, one can measure the packet train dispersion (i.e. the amount of time between the receipt of the last bit of the first packet and the last bit of the last packet). The model is implemented by cprobe [98], which is a non-cooperative tool. It sends a short stream of ICMP echo packets to the target host and assumes that it will respond by sending back ICMP echo-reply packets. Then the sender measures the dispersion of the ICMP Reply packets.

- *Probe Rate Model* (PRM) is based on the concept of self-induced congestion and treats the network as a queue with a service rate equal to the available bandwidth. If a source sends probes to a destination through the queue at a rate R less than A, probes will experience similar delays. On the other hand, if R is greater than A, probes will queue in the network and experience increasing delays. The model is thus based on the observation that the delays of successive probing packets increase when the probing rate exceeds the available bandwidth in the path, and consists in probing the network at different rates and detecting (at the destination) the point when delays start to increase. At this point, the probing rate is equal to the available bandwidth.

  This model is implemented by Pathload [101] and PathChirp [102], which are cooperative tools.

  Pathload introduces a technique based on Self Loading Periodic Stream (SLoPS): the algorithm consists of sending a stream of packets to the receiver. The receiver

measures the delay of each received packet and analyzes its variation. If the delay is constant, another stream is sent to the receiver at a greater rate. If the delay increases, another stream is then sent to the receiver at a rate between the two precedent values. This technique is repeated and the algorithm converges by dichotomy to the available bandwidth value. Due to its iterative nature, this algorithm can have long convergence times.

PathChirp proposes sending an exponentially spaced "chirp" probing train. The main advantage of this approach is to minimize the probing traffic load. Indeed, a single chirp is able to probe the network at different rates. Moreover, using a chirp of n packets allows pathChirp to exploit $n-1$ packet spacings that would require $2n-2$ packets using a packet-pair technique.

- *Probe Gap Model* (PGM), consists in capturing the relationship between the dispersion of a packet-pair and the rate of cross traffic at the bottleneck link of a path. It makes the same assumptions as PRM, and also assumes that the bottleneck link is both the tight link and the narrow link. This technique is implemented by Initial Gap Increasing (IGI) [103] and spruce [104], which are cooperative tools.

Bulk transfer capacity. The BTC (Bulk-Transfer-Capacity) represents the achievable throughput by a TCP connection on the end-to-end path. The IPPM proposes the following BTC definition: a measure of a network's ability to transfer significant quantities of data with a single congestion-aware transport connection. The intuitive definition of BTC is the expected long time average data rate of a single ideal TCP implementation over the path in question:

$$BTC = \frac{data\_sent}{elapsed\_time} \tag{2.1}$$

Since several TCP distributions (e.g. Tahoe, Reno, ...) implement in various way the congestion control algorithms, a BTC measurement methodology should theoretically define which TCP implementation it works for.

Metrics for evaluating broadband performance

Active monitoring broadband access networks is important to measure the contribution of the ISP network (i.e. the portion of the end to end path under the provider's control) to

the overall user experience, which depends on the composite effect of the segments their applications traverse end to end. Moreover, it enables the measurement of performance metrics necessary to establish Service Level Agreements for guaranteed and business class service offerings.

The DSL Forum proposed a management and monitoring architecture [31] and defined some additional methodologies and techniques purposely designed to evaluate performance on broadband access networks [16]. According to the specifications, ISPs can perform active tests between the Customer Premise Equipment (CPE) (i.e. the router) and a Network Test Server (NTS) located at their Points of Presence (POP) or in Internet (see Fig. 2.3). The measurements are managed by an Auto-Configuration Server (ACS) and can be both initiated by the CPE or by a NTS. In the latter case, for security reasons, the CPE protocol servers will respond only to the IP address of the NTS.

The foreseen diagnostics are based on standard TCP and UDP protocols and allow to measure the following metrics:

- *RTT, packet loss,* and *jitter* are obtained by utilizing an extended version of the UDP Echo service, called UDP Echo Plus, which adds some performance specific fields in the packet payload.

- *upload and download bulk transfer capacity* and *service response time* are obtained by performing an FTP or HTTP transaction to a corresponding remotely located FTP or HTTP server.

To perform the above mentioned measurements the following parameters are recorded:

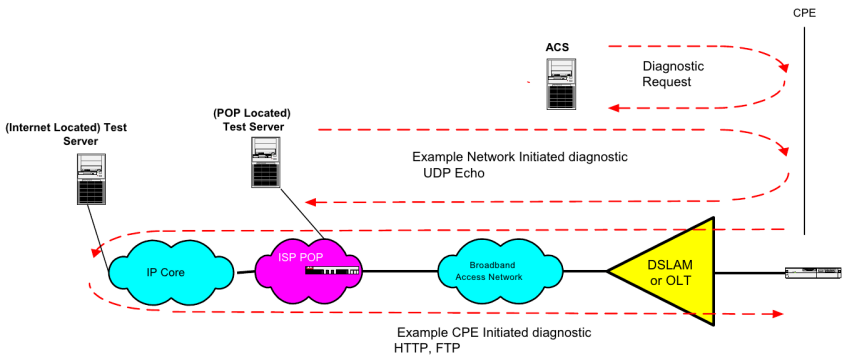


Figure 2.3: Performance testing components.

- *ROMTime*: Request time in UTC, which must be specified to microsecond precision. For HTTP this is the time at which the client sends the GET command, while for FTP this is the time at which the client sends the RTRV command.

- *BOMTime*: Begin of transmission time in UTC, which must be specified to microsecond precision For HTTP this is the time at which the first data packet is received, while for FTP this is the time at which the client receives the first data packet on the data connection.

- *EOMTime*: End of transmission in UTC, which must be specified to microsecond precision. For HTTP this is the time at which the last data packet is received, while for FTP this is the time at which the client receives the last packet on the data connection.

- *TestBytesReceived*: The test traffic received in bytes during the FTP/HTTP transaction including FTP/HTTP headers, between BOMTime and EOMTime.

- *TotalBytesReceived*: The total number of bytes received on the Interface between BOMTime and EOMTime.

- *TCPOpenRequestTime*: Request time in UTC, which must be specified to microsecond precision. For HTTP this is the time at which the TCP socket open (SYN) was sent for the HTTP connection, while for FTP this is the time at which the TCP socket open (SYN) was sent for the data connection.

- *TCPOpenResponseTime*: Response time in UTC, which must be specified to microsecond precision. For HTTP this is the time at which the TCP ACK to the socket opening the HTTP connection was received, while for FTP this is the time at which the TCP ACK to the socket opening the data connection was received.

- *PacketsReceived*: Incremented upon each valid UDP echo packet received.

- *PacketsResponded*: Incremented for each UDP echo response sent.

- *BytesReceived*: The number of UDP received bytes including payload and UDP header after the UDPEchoConfig is enabled.

- *BytesResponded*: The number of UDP responded bytes, including payload and UDP header sent after the UDPEchoConfig is enabled.
- *TimeFirstPacketReceived*: DTime in UTC, which must be specified to microsecond precision. The time that the server receives the first UDP echo packet after the UDPEchoConfig is enabled.
- *TimeLastPacketReceived*: Time in UTC, which must be specified to microsecond precision. The time that the server receives the most recent UDP echo packet.

In the following paragraphs we give more details on the methodologies proposed.

Download diagnostics utilizing FTP and HTTP transport. The Download diagnostic test is used to test the streaming capabilities and responses of the CPE and the WAN connection. The measurements are made during the download process, the files that are downloaded are arbitrary, and are only temporary. The file received is a stream of arbitrary bytes of a specified length, with no bound on the size.

When using FTP protocol, the server response to the FTP SIZE command gives the CPE the size of the file being Downloaded, while the response to the RTRV command will initiate the data sent on the data connection. To validate the measurement success, the CPE counts the bytes successfully received and compares their amount to the response to the SIZE command.

When using HTTP protocol, the server response to the HTTP GET includes the first TCP block of the file and either the HTTP header with the total file size or chunked encoding. The CPE counts the number of file bytes received successfully and is not required to retain the file in memory. Once the CPE has successfully received the number of file bytes specified in the HTTP response or chunked header the HTTP connection is closed. The CPE counts the number of bytes sent on the interface for the duration of the test.

Upload diagnostics utilizing FTP and HTTP transport. The Upload diagnostic test is used to test the streaming capabilities and responses of the CPE and the WAN connection. The measurements are made during the upload process, the files that are uploaded are arbitrary, without bound on size, and are only temporary. There are no storage requirements on the CPE for the uploaded files.

When using FTP protocol, the server response to the FTP STOR command gives the CPE a ready for transfer and it may begin the file transfer.The CPE counts the number of bytes sent successfully on the FTP data socket. The CPE is not required to retain the file in memory. To validate the measurement success, the CPE counts the bytes successfully received and compares their amount to the response to the SIZE command.

When using HTTP protocol, the server responds to the HTTP PUT with a successful response when the file has completed the upload, this will indicate a successful test. If the 200 OK is not received, or the TCP socket is torn down, the test will fail. The CPE may use chunked encoding and counts the number of bytes sent on the interface for the duration of the test.

UDP Echo Plus. UDP Echo Plus utilizes the UDP Echo Service and extends it by additional packet field definitions and new server behaviors and provides improved security. The ping tool has been widely used for realizing probing and general debugging in the IP context due to its ubiquitous availability in network routers and hosts. However, the viability of using ping measurements suffers from the fact that many routers process pings with lower priority than actual user packet's and may delay or discard ICMP echo requests in a manner that skews the measurement results. The UDP Echo Service bypasses this issue (unless explicitly port filtered at an intermediate or end host or router), because its packets traverse the same intermediate nodes and logical queuing paths as the user data traffic of the same class of service.

UDP Echo Plus is backwards compatible with UDP echo and devices capable of supporting UDP Echo Plus have no distinguishable effect on cooperating devices running standard UDP echo. However, when both cooperating devices are UDP Echo Plus capable, the utility of UDP echo is extended by the additional information provided in five new data fields:

- *TestGenSN* is the packet's sequence number set by the UDP client in the echo request packet, and is left unmodified in the response. It is set to an initial value upon the first requests and incremented by 1 for each echo request sent by the UDP client.

- *TestRespSN* is the UDP Echo server's count that is incremented upon each valid echo request packet it receives and responded to. This count is set to 0 when the

UDP Echo server is enabled.

- *TestRespRecvTimeStamp* is set by the UDP Echo Plus server to record the reception time of echo request packet and is sent back to the server in this data field of the echo response packet.

- *TestRespReplyTimeStamp* is set by the UDP Echo Plus server to record the forwarding time of the echo response packet.

- *TestRespReplyFailureCount* is set by the UDP Echo Plus server to record the number of locally dropped echo response packets. This count is incremented if a valid echo request packet is received but for some reason can not be responded to (e.g. due to local buffer overflow, CPU utilization, ...). It is a cumulative count with its current value placed in all request messages that are responded to. This count is set to 0 when the UDP Echo server is enabled.

By exploiting additional information, UDP Echo Plus is able to discern in which direction packet drops occur (i.e. one way packet loss per each direction) and packets discarded in the network from those discarded at the UDP Echo Plus server device, in addition to measuring one-way delay variation per each direction. Finally, in order to prevent a DoS (Denial of Service) attack on the CPE, the CPE will only respond to the UDP request from a predefined Source address on a configurable port number.

Final Remarks

We learned that many parameters should be taken into account to measure network performance and considering them at application layer allows to obtain results closer to the users' perspective. Moreover, as a result of our analysis of existing methodologies, techniques and tools to measure network performance, we conclude that there is no best approach for every context. Depending on the particular conditions in which measurements are conducted, different tools may be used to obtain the most accurate results.

2.2.2 Advertising broadband performance

ISPs usually advertise their broadband plans only in terms of download and (sometimes) upload speeds. As shown in Sec. 2.2.1, such metrics are not sufficient to describe the performance that user can expect when using various applications. When subscribing to

Table 2.2: Low level metrics for establishing VoIP quality.

VoIP Quality	Acceptable	Good
Latency	$< 300\ ms$	$< 150\ ms$
Jitter	$< 50\ ms$	$< 20\ ms$
Packet Loss	$< 3\%$	$< 1\%$

a connectivity plan, the user should be able to decide which offer better responds to his needs.

Network performance, like food, is characterized by many aspects which affect various applications in a different way. For instance, a heavy-downloader user would care more about higher throughput, while a user intrested in making VoIP calls, based on the requirements exposed in Tab. 2.2, would be more satisfied if experiencing low latency, jitter, and packet loss.

Driven by the requirement described above and by experimental results conducted on real home networks thanks to the BISmark platform (see Chapter 5), we contributed to the selection of a set of parameters to be presented to the user in a standard format. As defined in [34] such format, in analogy with "Nutrition Labels" for food items, reports both low-level parameters - represented as ranges of possible values - and high-level concepts (e.g. time to download a song, video streaming quality in terms of frames rate). We identify as its main properties the accuracy, the measurability, and the representativeness of applications performance.

The parameters we selected are explained in the following: *Sustainable throughput* and *Short-term throughput* relates to long transfers, while *Minimum throughput* is evaluated on high network load or congestion; *Baseline last-mile latency* and *Maximum last-mile latency* respectively represent the minimum and the maximum latency observed on the last-mile link; *Maximum jitter* represents the maximum degradation potentially affecting real-time applications; *Loss rate* and *Loss burst length* affect both throughput and real-time applications; *Availability* is the fraction of time the access link connection is established.

Particularly, we consider different throughput metrics due to the possible presence of traffic shaping techniques like *PowerBoost*^{TM} (see Sec. 1.1.1), which makes short throughput measurements achieve higher values than they can obtain in the long term. Moreover, we focus latency measurements on the last-mile link, since it gives the higher contribu-

Network Metric	**WiMAX user** 6 Mbps down 1 Mbps up	**DSL user 1** 6 Mbps down 0.50 Mbps up	**DSL user 2** 3 Mbps down 0.38 Mbps up	**DSL user 3** 3 Mbps down 0.38 Mbps up	**Cable user 1** 22 Mbps down 4 Mbps up	**Cable user 2** 22 Mbps down 4 Mbps up
Sustainable throughput (Mbps)	6.0 down 1.0 up	6.0 down 0.50 up	3.0 down 0.38 up	3.0 down 0.38 up	**12.5 down** 2.0 **up**	**12.5 down** **2.0 up**
Short-term throughput (Mbps)	6.0 down 1.0 up	6.0 down 0.50 up	3.0 down 0.38 up	3.0 down 0.38 up	**22 down for 8s** **3.8 up for 20s**	**18.5 down for 13s** **7.0 up for 8s**
Minimum throughput (Mbps)	6.0 down 1.0 up	6.0 down 0.50 up	3.0 down 0.38 up	3.0 down 0.38 up	12.5 down 2.0 up	12.5 down 2.0 up
Baseline last-mile latency (ms)	**65**	8	8	24	8	8
Maximum last-mile latency (ms)	700	800	**2000**	**2000**	300	750
Loss rate (%)	0.03	0.02	0.05	0.01	0.08	0.01
Loss burst length	—	—	—	—	—	—
Jitter (ms)	**8.7**	2.4	1.6	2.2	1.4	2.1
Availaility (%)	95.6	95.2	94.8	94.0	94.7	96.6

Table 2.3: Labels applied to six broadband connections from different ISPs and service plans.

tion when evaluating the same metric end-to-end. In addition to these metrics, the label should also provide information on specific QoS policies adopted by the ISP, to inform the user about their presence.

We applied the label to six broadband connections across three different ISPs in the USA, by considering the measurements taken with BISmark for the considered metrics over one month (see Tab. 2.3). By considering the bold values in table, we demonstrate of how the label could help in distinguish Internet service plans having the same advertised characteristics. Indeed, as best shown in Fig. 2.4 users having different access technologies experience significantly different latencies.

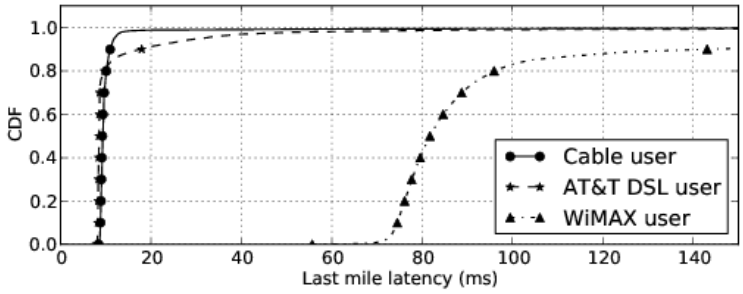


Figure 2.4: Latency profile for Cable, DSL and WiMAX.

Chapter 3

Evaluating the performance of broadband access networks

Focusing our attention on broadband access networks, we performed an extensive study of existing projects that address access networks performance evaluation. In the following sections we first present a taxonomy for broadband benchmarking approaches and then we define an architecture for benchmarking access networks from the edge.

3.1 A taxonomy for broadband benchmarking approaches

Based on our study of projects addressing the evaluation of access network performance, we define a taxonomy dividing all the existing approaches into three macro-categories depending on the location of the vantage points used to perform the measurements. As shown in Fig. 3.1, such macro-categories are then further divided in categories and sub-categories in order to properly distinguish all the significant differences among the existing approaches.

Figure 3.1: Taxonomy for broadband benchmarking approaches.

3.1.1 Server-based approach

We first consider the macro-category in which measurements are conducted from servers in the core of the network towards users' networks (see Fig. 3.2). In general, all the approaches which make use of vantage points located outside the users' networks have limitations, mainly because of the difficulty in accounting for confounding factors in the user's home network. Indeed, in most cases users' networks are connected to the Internet through a NAT, which hides its potential complex structure behind a single IP address. Also, most ISPs utilize dynamic IP addressing for their clients, thus making difficult to measure the same access link repeatedly over time.

The server-based macro-category can be further divided into three categories, as described in the following sections.

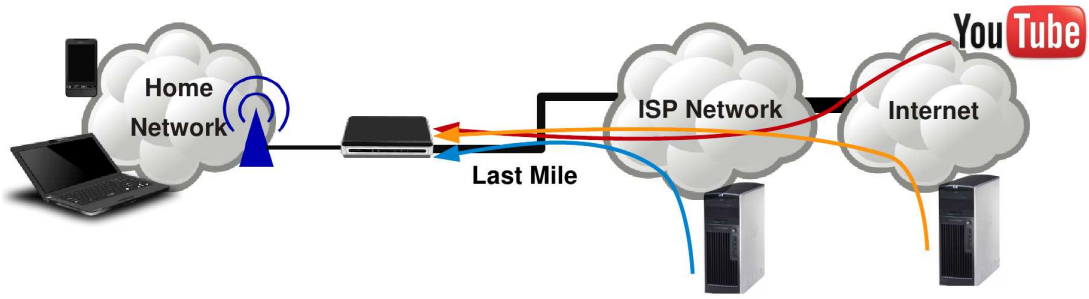


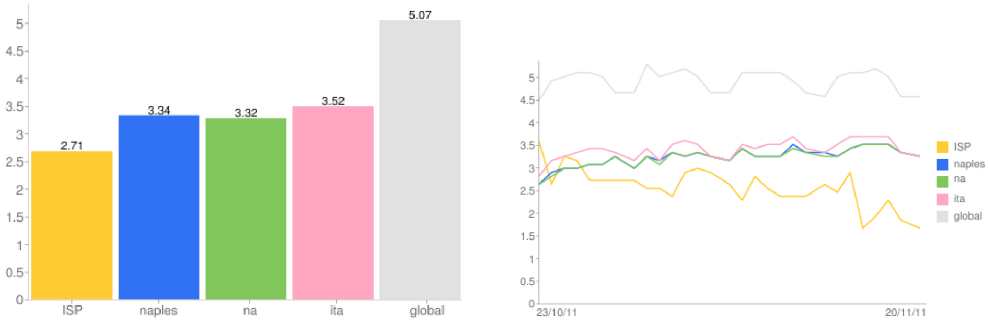
Figure 3.2: Server-based approach.

Service-based approach

We classify as service-based the approaches in which passive measurements are performed from servers directly offering a specific service.

The most famous example is the Youtube Speed Meter [36]. This service shows the history of the registered speeds of YouTube videos watched from users' location (i.e. a browser associated with a specific IP address). It also shows aggregate video speed by city, state, country, and worldwide. The methodology adopted to compute the speeds is the following.

They first compute the bandwidth of almost every YouTube played video by considering the amount of data sent to and acknowledged by the user's machine in a given time period. Small video responses are excluded as they can add noise to the bandwidth calculation. The estimated download bandwidth of every video played is associated with



(a) Average bandwidth (Mbps). (b) Day-by-day bandwidth (Mbps).

Figure 3.3: Sample bandwidth reported by Youtube Speed Meter.

a `VISITOR_INFO1_LIVE` cookie and with the IP address that requested the video. For the daily speed value, as shown in the YouTube speed page, they average the bandwidth for all videos played from a given cookie and IP address during the course of the day. If a single laptop user uses multiple network connections and watches videos regularly from these, they could see different speed measurements and historical data based on which connection is currently in use. Per-ISP measurements are computed by averaging the daily averages of all users who use the same ISP and are in the same geographic region. The ISP speed value is an average of all users across all types of Internet connectivity and service plans offered by the ISP. There are many factors affecting the measurement conducted, like the ISP utilized, the distance to Google servers, the computer used, and other devices in the user's network such as other computers and Internet connected appliances. This approach is not very accurate in calculating the downstream bandwidth, because it is computed on the sender side by indirectly relying on acknowledgments, which may follow a different path and can be affected by the narrow upstream bandwidth.

Core-based approach

We identify as core-based the approaches which conduct active measurements from dedicated servers. Some studies have characterized access network performance by probing access links from servers in the wide area [7, 37]. Active probing from a fixed set of servers can characterize many access links because each link can be measured from the same server. Unfortunately, because the server is often located far from the access network, the measurements may be inaccurate or inconsistent. Isolating the performance of the access network from the performance of the end-to-end path can be challenging, and

dynamic IP addressing can make it difficult to determine whether repeated measurements of the same IP address are related to the same access link over time. A remote server also cannot isolate confounding factors, such as whether the user's own traffic is affecting the access-link performance.

ISP-based approach

We classify as ISP-based the approaches providing passive measurements conducted by the ISP itself. Previous work has characterized access networks using passive traffic measurements from DSL provider networks in Japan [38], France [39], and Europe [9]. These studies mostly focus on traffic patterns and application usage, but they also infer the round-trip time and throughput of residential users. However, without active measurements or a vantage point within the home network, it is not possible to measure the actual performance that users receive from their ISPs, since the user traffic does not always saturate the user's access network connection. For example, Siekkinen *et al.* [39] show that applications (e.g. peer-to-peer file sharing applications) often rate limit themselves such that performance observed through passive traffic analysis may reflect application rate limiting, as opposed to the performance of the access link.

3.1.2 Host-based approach

The second macro-category we consider is the one in which passive or active measurements are performed from users' devices (e.g. personal computers, laptops, mobile phones, tablets, ...), as shown in Fig. 3.4. In general, utilizing a vantage point inside the user's network provides the advantage of having the same perspective as the user and to easily scale to a large number of installations. By exploiting users' devices, they can usually rely on high computational resources (e.g. CPU, memory) and large storage capabilities. On the other hand, they have some limitations:

- they cannot consider cross-traffic in the local network while performing measurements;
- they can suffer from computational bottlenecks from other applications in the user's device;
- since user devices are rarely switched on all the time, they are not continuously available.

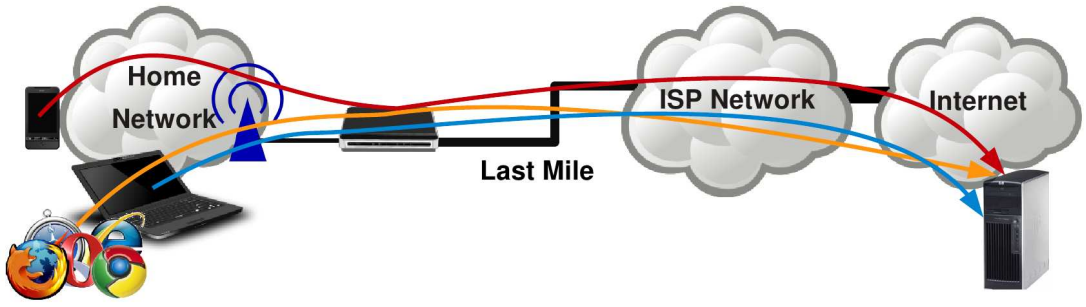


Figure 3.4: Host-based approach.

The host-based macro-category can be further divided into two categories, as described in the following sections.

Client-based approach

We classify as client-based the approaches requiring the installation of a stand-alone software client on users' personal computer.

This approach has been recently adopted by many projects devoted to measure broadband access links performance. We analyze the most relevant.

Grenouille project [40]. In 2000, a group of volunteers started the Grenouille project to monitor the performance of residential access providers in France. This project now has thousands of members across all major French cities and Internet service providers. To participate, users download the Grenouille client, which runs periodic tests to measure their provider's performance. The Grenouille client runs on Windows, MacOS, and Linux. The client performs three types of periodic measurements: round-trip time (RTT), average FTP download rate, and average FTP upload rate. After collecting these statistics, the client sends the results to a central server. The server aggregates these measurements to aggregate statistics for each ISP, SLA, and city. Users can view the performance statistics for each ISP at www.grenouille.com.

To configure the client, users create an account with Grenouille providing information about their connection, the city where they live, their ISP, and their SLA. Based on these parameters, the Grenouille server configures the measurements that the client should perform. The destination for both the RTT probes and the FTP uploads is always a Grenouille server (except for *Numericable*, which has its own upload hosts). The source of the FTP download depends on the ISP. The client usually downloads the files from a

server inside the user's own ISP. If the ISP does not have an FTP download server, then clients download the file from the Grenouille server. The size of the file varies according to the SLA so that the client does not congest the network for users with slower connections.

Each Grenouille client performs measurements using ping and FTP; the client first checks that the network card is idle before performing measurements. This minimizes interference with traffic or activities at the end host that might affect the measurements. Every 30 minutes, a client performs one FTP download and one upload to estimate download and upload speeds and estimates the RTT and loss rate with the average of ten consecutive pings. Clients periodically send the result of these measurements to the server along with the time elapsed between the end of the performed measurements and the time in which the client sends the report to the server. The server then timestamps the measurement by subtracting this time difference from the time at which it receives the message. This mechanism allows the server to synchronize the measurements from the clients without requiring the clients' clocks to be synchronized. The server then truncates the resulting timestamps to the nearest 30-minute timestamp and averages the performance measurements it received over that period from the client. The average values are finally stored in a database.

NETI@Home project [41]. NETI@home is an open-source software (now inactive) designed to collect network performance statistics from end-systems. It is written in C++ and tested on the Windows, Solaris, and Linux operating systems. It is designed to run on end-user machines and collects various statistics about Internet performance. Its basic approach is to sniff packets sent from and received by the host to infer performance metrics. By using an external component, the packets are sniffed without setting the network interface in promiscuous mode and are then sent to NETI@home, where they are sorted into bidirectional flows to be analyzed. Once a specified number of packets are analyzed, the data is compressed and sent to a server at the Georgia Institute of Technology to be stored into a publicly accessible database. Researchers are then able to sort the data via a Web interface to look for specific data and/or trends.

In order to protect users' privacy NETI@home provides the opportunity to select a privacy level that determines what types of data are gathered, and what is not reported. At the highest privacy setting, no IP addresses are recorded, while at the lowest privacy setting, all IP addresses are recorded.

Mark-and-Sweep project [42]. Han *et al.* measured access network performance from a laptop by exploiting open wireless networks. The method adopted to collect network information, called mark-and-sweep, focuses on maximizing the amount of collected data for the amount of human time invested in taking measurements. It utilizes a two-pass measurement scheme. In the first pass, they drive through an area without stopping, collecting extensive passive measurements of all access points without associating with them. These measurements are then used to create a plan for the second pass, in which they measure once each access point and the associated network at the approximate location where its signal strength was the strongest. Once it obtains an IP address on the target wireless LAN, the tool performs five tests:

- pings a test server in the CMCL lab at Carnegie Mellon University;
- attempts to open a TCP connection to the test server on five ports: 25 (SMTP), 80 (HTTP),443 (HTTPS), 587 (authenticated SMTP) and 56123 (a high-numbered port);
- runs a traceroute towards the test server;
- uses the open source STUN client/server to determine the type of NAT.
- sends UDP packets at 15 *Mbps* for 4 seconds to measure upstream and downstream bandwidth by adopting a modified version of nuttcp [105] able to work through NATs.

The unusual approach utilized is convenient, because it does not require user collaboration. However it does not scale to a large number of access networks, it cannot collect continuous measurements, and it offers no insights into the specifics of the home network configuration. Moreover, relying on wireless connectivity the accuracy of measurements, performed once for each LAN, may be highly compromised.

BSense project [106]. The BSense project integrates both estimated and measured broadband data and combines it to demographic and geographic data in order to generate broadband maps. A set of Web services APIs can be used to feed the system with initial coverage estimations and broadband measurements. The gathered information is stored in a relational database enabled for spatial information storage and it is then processed via a data fusion engine, which enables automated and evolving broadband census creation.

Measurements are conducted using a software multi-platform client, written in C++, and installed by users on their personal computers. During the first startup the client prompts the user for a valid postcode in Scotland, then asks four basic questions on access technology type, ISP name, and advertised speeds for the service purchased by the user. During the normal operation, relying on the D-ITG tool [107], the client measures the following parameters: upstrean and downstream throughput, latency, and packet loss.

Isposure project [43]. *Between* is a trusted advisor to the Italian government on national broadband performance and publishes regular reports through its "Osservatorio Banda Larga" portal, which informs consumers, business and the ISP industry about issues pertaining to broadband services across Italy. Between has launched the Isposure broadband analysis service which consumers can use to test the performance of their Internet connection. The Isposure client is a freely-downloadable Windows application which performs the following tests on users' connection: download and upload line speed, Web browsing speeds, gaming speed (i.e. RTT), and DNS lookup times. The measurements are conducted when no cross-traffic is detected, towards the closest server selected among the three located in San Francisco, Amsterdam, and Brisbane. Each user can access measurement results from the Isposure Web portal, where per-measurement average values are shown over configurable time periods (i.e. week, month) on simple plots.

Neubot project [44]. The Neubot project, born from the collaboration between NEXA Center for Internet & Society and the DAUIN group of University of Torino, provides a multi-platform client, written in python, available for Ubuntu Linux, Windows XP and Mac OSX. It is executed in background on user's workstations and offers a local Web server which provides an interface to interact with it. Every 30 minutes, if no other traffic is generated by the host, the client performs measurements, using both HTTP and BitTorrent protocols, of the following parameters: round-trip time, upload, and download goodput (i.e. the application level throughput). Such measurements are conducted towards a servers hosted in the city of Trento. The results are accessible to the user through the local Web interface, where per-measurement average values are shown over time on simple plots. All the results are also sent to a central server, where they are stored for further processing.

Table 3.1: Statistics reported by the Ne.Me.Sys. certificate.

Statistic	Comments
Maximum download bandwidth	95^{th} percentile
Minimum download bandwidth	5^{th} percentile
Average download bandwidth	-
Download bandwidth standard deviation	-
Maximum upload bandwidth	95^{th} percentile
Minimum upload bandwidth	5^{th} percentile
Average upload bandwidth	-
Upload bandwidth standard deviation	-
Download failure rate	-
Upload failure rate	-
Average one-way transmission delay	$RTT/2$
One-way transmission delay standard deviation	-
Packet loss probability	ICMP

Ne.Me.Sys. project [45]. The italian authority for telecommunications (AGCOM) approved Ne.Me.Sys., developed by Ugo Bordoni Foundation, as the official software to measure quality of broadband Internet connections, According to [14], it allows to evaluate the quality of a broadband connection thanks to a set of measurements customized depending on the ISP utilized by the user. The measurements are conducted by a software client installed on the user's personal computer over a variable time period (from 24 to 72 hours). During such period, the user is asked to not interfere with client operation in any way (e.g. utilizing other networked applications) and the client avoids starting measurements if other traffic is being generated. The measurements are made towards two servers located in a so called "Neutral Access Points" located in Rome, which correspond to an Internet exchange point and represents the closest network neutral with respect to specific ISPs. The client performs the following measurements: upload and download speed using FTP , and one-way delay and packet loss using ping. The results are then sent to a central server which extracts a long list of statistics (see Tab. 3.1), which is exported in PDF format to certificate perceived performance.

Web-based approach

We identify as Web-based the approaches which require the execution of a software component inside the Web browser. In general, by just requiring a Web browser, this approach allows to easily conduct measurements on a large scale. On the other hand, working in that context, it is difficult for a measurement tool to be accurate, due to the additional

software layers between the tool and the physical devices (e.g. the network interface).

The Web-based category can be further divided into two categories, as reported in the following paragraphs.

Plugin-based approach The plugin-based approach requires a browser plugin to be installed into the browser, which represents the vantage point from which the measurements are conducted. This approach has been mostly adopted to perform Web performance troubleshooting, which involves measurement of parameters at layers higher then transport (e.g. application layer).

For instance Cui *et al.* [46] (Fig. 3.5) designed a troubleshooting setup based on Firefox and composed of three parts: a plugin for the Web browser, a packet-level capture and a database repository.

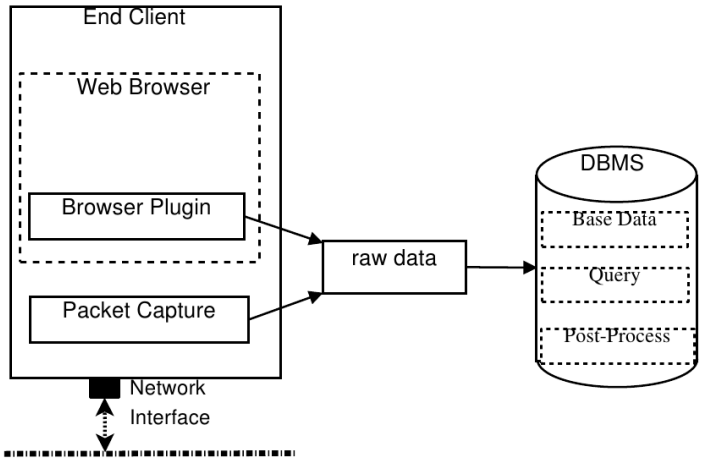


Figure 3.5: Troubleshooting setup architecture.

The browser plugin tracks some critical events during a Web session that are related to the user's perceived experience. It tracks paint events to measure the waiting time elapsed between the user clicking on a Web page and the content of the Web page being displayed. The plugin adds some event-listeners inside the browser:

- the *first painting event* tells how long the user needs to wait for the first visual impact. They define the *first impression time* as the time interval between the user clicking on the URL and the first painting event.

- the *full page load event* measures how long it takes until the entire content of a Web page, which may consist of several tens of different elements, is displayed. They

define the *full load time* as the time interval between the user clicking on the URL and the full page load event.

For each Web element that is loaded, the plugin records a certain number of additional information (e.g. URI, ...).

This approach has much in common with the client-based one, since browser plugins[1] mostly allow to implement the same measurements as stand-alone applications. On the other hand they differ in their availability and operating software layer. Indeed, in the first case, the tool is active only when the browser is opened. In the second case, as the plugin is usually spawned as a thread by the browser process, the scheduling algorithm operating on the measurements may be different and compete first with other browser threads and then with other processes running on the host.

Embedded approach We classify as embedded the approaches which include measurement tools into a container appearing on a Web page (e.g. *Flash*, *Java*, *Silverlight* ...).

Most projects and studies adopting this approach typically help users to troubleshoot performance problems by asking them to run tests from a Web site and running analysis based on these tests. The most famous is SpeedTest.net [47], which offers a fancy Flash interface (shown in Fig.3.6) to measure latency, and upload and download speed.

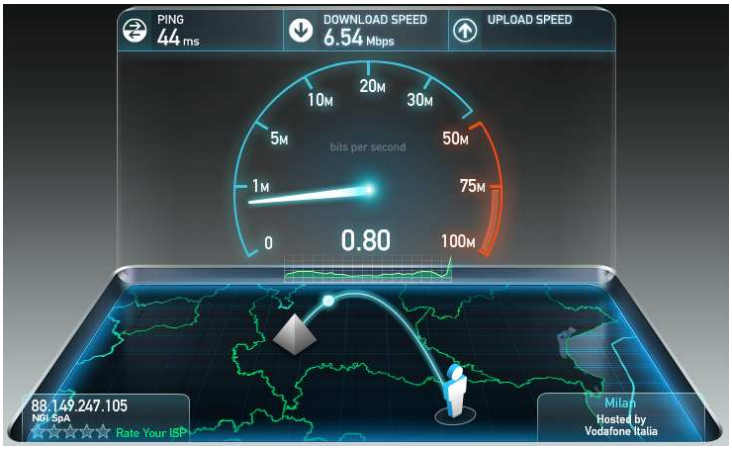


Figure 3.6: SpeedTest.net fancy interface.

Netalyzr [8] measures the performance of commonly used protocols using a Java applet that is launched from the client's browser. Network Diagnostic Tool (NDT) [5] and

[1]Browser plugins, unlike extensions, are usually not limited by restrictions usually applied to javascript code.

Network Path and Application Diagnostics (NPAD) [4] send active probes to detect issues with client performance. Glasnost performs active measurements to determine whether the user's ISP is actively blocking BitTorrent traffic [48].

Users typically run these tools only once (or, at most, a few times), therefore the resulting datasets cannot capture a longitudinal view of the performance of any single access link. Moreover, in this case, there are more additional software layers between the tool and the physical devices, since the measurements are conducted from inside a container in the Web page. Even worse, the measurement tools are implemented inside environments like Java or Flash, which significantly impact their accuracy.

3.1.3 Router-based approach

We consider as third macro-category the one in which both active and passive measurements are conducted directly from the router acting as gateway to the Internet for the local network (see Fig. 3.7). In general, adopting such a vantage point solves most of the issues encountered by all the other approaches.

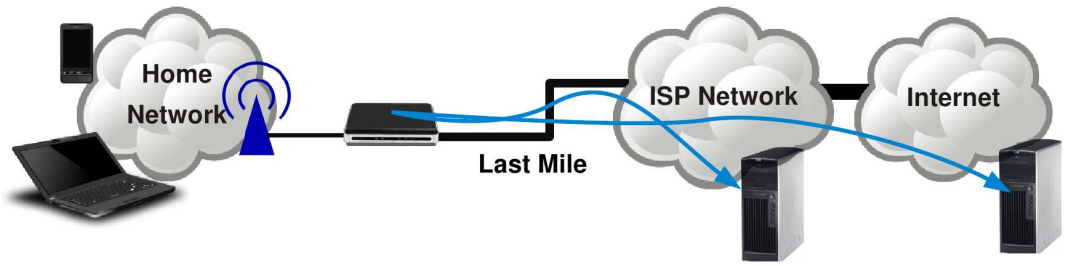


Figure 3.7: Router-based approach.

Indeed, exploiting the home gateway offers the following advantages:

- the router is switched on all day, so it solves the availability problem, thus allowing to perform measurements at any time and at regular schedules and to perform continuous passive measurements;

- because the router is involved in any communication between the local network and the Internet, it can observe any kind of cross-traffic, thus allowing to conduct active measurements only in positive conditions or to properly correct/explain their results in all the circumstances;

- the router is a completely controlled environment, so inter-process interference can be avoided during measurements.

On the other end this approach has a few drawbacks:

- obtaining a large scale deployment is not an easy task, because the only choices are to ship preconfigured gateways, which is costly, or to request users to upgrade the firmware of their router, which requires medium-skilled users and a preexisting compatible router;

- most routers offer low computational resources (i.e. CPU and memory) and small storage capabilities (e.g. 4 MB flash drives), which makes challenging the implementation and the management of measurement tools.

The only existing project adopting this approach is SamKnows [49], which has been funded by UK and US governments to conduct broadband studies in their countries respectively under the direction of Ofcom (Office of Communications) and FCC (Federal Communication Commission), and has now contracted with the European Community for a similar study in Europe. SamKnows deploys gateways in each participant's home either directly behind the home user's modem or behind the home wireless router. The gateway is a Netgear WNR3500L RangeMax Wireless-N Gigabit router with a 480 MHz MIPS processor, 8 MB of flash storage, and 64 MB of RAM as well as five 10/100/1000 auto-sensing Ethernet ports and 802.11n Draft 2.0 WiFi. It can be updated and managed remotely and periodically executes active measurements to estimate the parameters listed in Tab. 3.2.

3.1.4 Comparison of approaches

The analysis conducted on all the approaches of the proposed taxonomy brought us to the following conclusions.

There is no perfect approach to address broadband access network performance evaluation, since all the approaches have advantages and drawbacks. Among all of them, the most promising approaches are those working from the users' side, because they can better take into account the context in which measurements are performed. We identify router-based, client-based and plugin-based as the most promising approaches, since they allow to perform experiments on the same access link scheduled over a long time period.

Table 3.2: Tests currently performed by the SamKnows router.

Parameter	Interval
Multi-threaded HTTP download speed	2 hours
Multi-threaded HTTP based upload speed	2 hours
Availability of the connection	30 sec
Jitter	1 hour
Latency (ICMP)	12 min
Latency (UDP)	10 sec
Packet loss (ICMP)	12 min
Packet loss (UDP)	10 sec
DNS query resolution time	1 hour
DNS query failure rate	1 hour
Web page loading time	1 hour
Web page loading failure rate	1 hour
Video streaming performance	2 hours

In other words, they allow to repeat the same experiment many times, thus obtaining more statistically significant results.

With respect to the measurements conducted, as also noticed by [35], we found that each existing project adopts different methodologies, techniques, and tools to measure a similar set of parameters. Moreover, many of them refer to the same parameters with different names and none of them consider all the performance metrics we selected in Sec. 2.2.2. To highlight this aspect we report in Tab. 3.3 a comparison of four client-based projects on all the essential properties.

As shown they are implemented using different programming languages and not all of them are open-source, thus making impossible to verify their accuracy. They mostly measure different aspects of throughput, latency, and packet loss by adopting different protocols and methodologies. They measure throughput by transferring files having a fixed size, which may be not optimal for every access network, since on high speed links the experiment may have a too short duration[2]. They mostly provide average values, which may not be accurate enough. Most of them provide a mechanism to avoid measurements when other applications are generating traffic, and directly ask the user for information about his geographic location and subscribed broadband plan.

[2]TCP-based measurements should last at least 15 seconds to allow the slow start phase to finish.

Table 3.3: Comparison of four client-based projects.

	Ne.Me.Sys.	NeuBot	Isposure	BSense
OpenSource	yes	yes	no	yes
Language	python	python	Visual C++	C++
Multi-platform client	yes	yes	no	yes
Throughput	FTP (1 MB file)	HTTP multi-thread (50 MB file up) (90 MB file down), BitTorrent (64 $KB - 1$ MB file)	TCP multi-thread on port 80 (15sec)	TCP and UDP (15sec)
Latency	ping (1 pps - 10 sec)	ping (1 pps - 10 sec)	ping (1 pps - 10 sec)	UDP (10 pps - 60 sec)
Jitter	-	-	-	-
Packet loss	ping (1 pps - 10 sec)	-	-	UDP (10 pps - 60 sec)
Browsing experience	-	-	HTTP (top 5 sites)	-
DNS delay	-	-	top 10 sites	-
Failure rate	yes	no	no	no
Detail of results	5^{th} & 95^{th} percentile, mean, standard deviation	mean	mean	samples per tenth of a second
Measurements interval	-	30 min	24 hours	30 min
Cross-traffic check	yes	yes	yes	no
Geolocation	user-aided	-	user-aided	user-aided
Measurements negotiation	n.d.	yes	n.d.	no

3.2 An architecture for benchmarking access networks

As a result of the analysis conducted in the previous section, we identified the characteristics an ideal architecture, able to measure broadband access networks on a large scale, should have. We report in the following sections an overview of such architecture and a brief list of all the features which can be important to allow its wide deployment.

3.2.1 Architecture overview

In Sec. 3.1 we concluded that the only approaches able to take into account the context in which measurements are conducted and to perform experiments on the same access link properly scheduled over a long time period are the router-, client- and plugin-based.

Hence, we defined an architecture which can work with all such approaches, coping with all the challenges related to a wide distributed architecture.

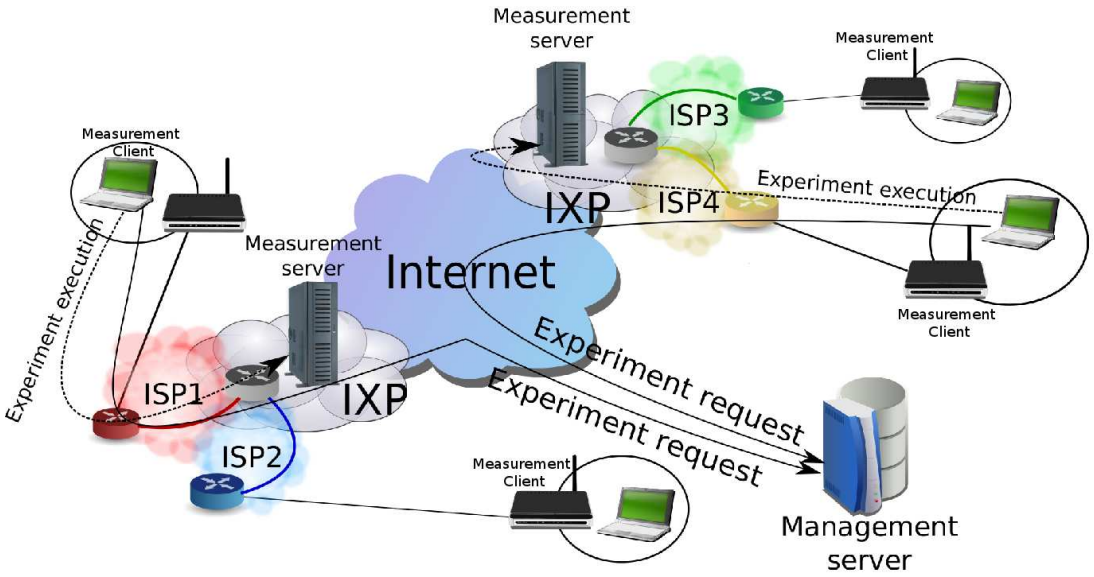


Figure 3.8: Overview of an ideal architecture for benchmarking access networks.

An overview of such architecture is shown in Fig. 3.8. It is made of three essential components:

- *Measurement Client*: installed on a device part of the users' network (e.g. personal computers, laptops, mobile phones, tablets, routers, access points, set-top-boxes ...), it periodically requests for instructions on experiments to execute and reports back the obtained results;
- *Measurement Server*: deployed on a network as close as possible (in network terms) to the clients and provided with a high capacity connection to the Internet, it acts as source/destination for active measurements requiring control on both sender an receiver side;
- *Management Server*: equipped with high computational resources and storage capabilities, as well as high capacity link to the Internet, it is responsible for managing all the experiments conducted by measurement clients and to collect and properly organize their results.

The management server, which may be clustered or properly distributed to avoid single-point-of-failure problems, is responsible for the following tasks:

- monitoring the correct operating state of measurement clients over time;
- managing measurement clients software updates both manually and automatically;
- configuring and planing experiments to be performed by measurement clients, in order to avoid significant interferences among them and to balance their load on different measurement servers;
- collecting experiments' results and properly organizing them in a relational database, in order to allow the extraction of statistics at different aggregation levels.

Measurement clients, in the best case, should cover all the interested geographic locations (preferably more clients for each location having different ISPs and plans), and the measurement servers should be distributed to have at least one of them at the shorter possible network distance (their number should also be proportional to the number of clients in a given location).

In addition to these basic components, such architecture may provide some other optional components which can act to provide incentives for users to be involved in the experiments. Indeed, users would be happier to participate if the results of conducted experiments are shown through easily understandable and fancy graphical representations. To accomplish this goal, the architecture may also include the following components:

- *Front-end Web Server*: it offers accurate and detailed graphical representations of statistics both related to single users' (through a login process) and aggregated at different layers (publicly accessible), allowing also to compare results among different ISPs and plans in the same geographic area.
- *Mapping Server*: it produces geographical representations of extracted statistics to allow a better understanding of performance on a wide area.

3.2.2 Important features the architecture should provide

In the following we briefly outline the features we believe are essential for an ideal architecture as the one defined in the previous section:

- measurement clients and servers should support pre-existing measurement tools, which are commonly well tested and thus more accurate;

- the management server should be able to allocate resources for measurement servers without the need to communicate with them, in order to allow also measurements towards servers not part of the controlled architecture (e.g. servers offering real services);

- all the basic components should provide a flexible support for underlying measurement tools, in order to allow a proper configuration of their parameters depending on the context in which they operate;

- with respect to a measured access links, the basic components should be able to automatically detect and track both its geographical position and ISP service plan, in order to obtain such information without relying on users' cooperation (which may lead to unverifiable mistakes).

Chapter 4

HoBBIT: adopting the client-based approach

After defining the guidelines reported in Sec. 3.2, in order to easily reach a large scale deployment and to evaluate the performance of broadband access networks at higher geographical resolution, we decided to build a client-based platform.

Hence, we designed and developed the HoBBIT [1] (Host Based Broadband Internet Telemetry) platform, which is specifically targeted to evaluate the performance of broadband access networks in Italy.

It provides a multi-platform client application implementing only active measurements and offers, a as an incentive for the user, a fancy front-end web interface through which he can visualize all the statistics extracted about his access network up to the detail of a single experiment. Moreover, the interface gives also the possibility to visualize statistics at different aggregation levels. For instance, it is possible to visualize the statistics per geographic area on fancy interactive map.

In this chapter we first describe the design and the implementation of the components part of the architecture, and then the experiments we are currently performing. Finally we give some insight on the challenges we had to face and on preliminary results we obtained by analyzing collected data.

4.1 Architecture components

The architecture designed for HoBBIT includes all the basic components required by the ideal architecture defined in Sec. 3.2: *measurement clients* (henceforth clients), *measure-*

[1] http://hobbit.comics.unina.it

ment servers, *management server*, *front-end server* and *map server*.

The behavior of HoBBIT clients, which can move among different access networks or may operate together in the same access network, makes it challenging to properly track their association with a specific access network. This is primarily true for mobile computers (e.g. laptops, netbooks, mobile phones, tablets, ...), which are today really common. Therefore, we had to adopt specific solutions to cope with it.

Moreover, we designed the HoBBIT framework to give a high flexibility in the definition and execution of the experiments and in their assignment to clients.

In the following sections we describe the design and the implementation of the components of the architecture.

4.1.1 The management server

In this section we provide a detailed description of the design made for the management server giving also some insights on the choices adopted to cope with the mobility of the clients and to maintain a good flexibility for the definition of the experiments.

Requirements analysis and main concepts

The Internet today is an extremely complex environment subject to continuous changes. The HoBBIT project focuses on that set of links which are subject to commercial contracts, being therefore mainly of interest for the final users. These links are commonly identified as the *last miles*, since they connect the user with the first telephone box towards Internet.

The users of the HoBBIT project range from home users to big corporations and aim at monitoring the owned network and its connectivity. Since HoBBIT users, by moving among different access networks, can perform measurements on different access networks, all the measurements performed by the platform should be maintained by using an access network centric approach.

The platform should require information about the provider and the contract which regulates each connection with its nominal values. Each connection should then be measured both to identify connectivity problems and to collect statistical data. Such measurements should usually follow a predefined schedule in which their duration and the possible number of repetitions is stated. Furthermore, measurements should be be part of wider campaign of measurements according to a specific objective.

In addition, the platform should take advantage of the geographical position of each connection (e.g. the municipality), pursuing the goal of collecting measurements and correlating their results also with the geographical location.

Hence, each installation of the HoBBIT client should perform several measurements according to the schedule associated to the connection currently in use. The results of the measurements performed by a specific installation should then be accessible only to the owner user. Logging into the system with username (i.e. the email address) and password, he should be able to access to graphical representations and synthetic statistics about his connections. Moreover, users should be able to visualize measurements results aggregated per geographical area (e.g. region, province, municipality, ...) and service plan details (ISP, access tehcnology, ...).

Broadband connection. The main goal of the project is to obtain statistics about broadband access networks (henceforth connections) at different granularities. We define the concept of connection through the following reasoning. Since the object of the measurements are the connections (i.e. the last-mile links between home routers and ISPs), it is possible to detect if different users access to the Internet through the same connection by comparing the MAC address of their default gateway. However, in some cases such MAC address is not sufficient. In the example reported in Fig 4.1, in which client 3 and 4 are linked to different interfaces of the same router (i.e. wired and wireless interfaces), we cannot determine them as utilizing the same connection just relying on the gateway's MAC address, because they see different gateway's MAC addresses. To cope also with such case, we take into account also their public IP address, which is the same for clients 1 and 2, as well as for 3 and 4. Thus, HoBBIT identifies two clients on the same connection if they perceive the same gateway's MAC address or if they report the same public IP address enough close in time. Thus, by aggregating clients per connection, the platform can coordinate them to cooperate for pursuing the common goal of evaluating its performance.

Client installation. The HoBBIT client can be installed on devices having at least a supported operating system (i.e. Microsoft Windows, Linux and Mac OSX) and a connection to the Internet. Each installation is identified by an UUID (Universally Unique Identifier) assigned from the management server at registration time. We give more details

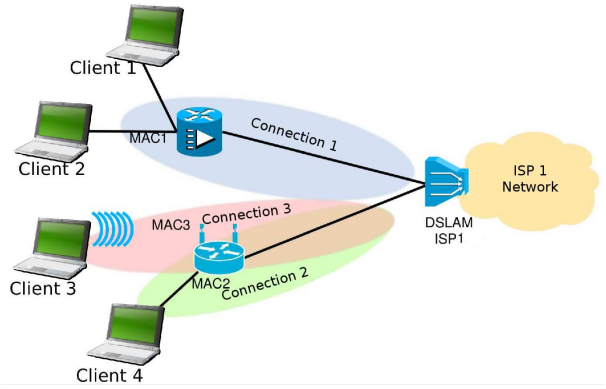


Figure 4.1: Example of connection as seen by HoBBIT .

on measurement clients in a following section.

Measurement experiment. A measurement experiment (henceforth experiment) aims at measuring specific connection parameters. While the selected parameters depend on the goal of the experiment, the act of measuring depends on the connection. For this reason, a measurement experiment is totally defined by a set of parameters and a set of rules used to measure those parameters. In this way, from the combination of these two factors, each measurement experiment may produce one or more results. For example, with a single experiment, by properly configuring parameters and rules, we can estimate jitter, latency and packet loss.

The platform contemplates experiments classification based on intensity (invasive or light), type (e.g. network neutrality assessment), and so on, such that it is possible to easily distinguish and aggregate experiments and their results per class.

The term parameters refer to a list of options which allows to identify an experiment, while the rules are the values associated to these options. Sometimes, such rules are not statically defined from the system administrator during the experiment definition: in this case, the server elaborates the incoming explicit request for it and provides the required answer to client. Indeed, the rule values depend on a high number of factors most of the cases not predictable. A classic example is an experiment which requires the IP address of the measurement server. This choice cannot be provided by the administrator since it depends on the measurement servers: statically assigning experiments to measurement servers may be hazardous. For this reason, the client requests form the management server the IP address to consider for the measurements. Such address is dynamically determined

form the server by carefully analyzing the current measurement servers availability.

Finally, during the experiment creation, the administrator defines a `bash` script that will be executed by the involved clients.

Measurement campaign. In order to collect meaningful statistical information about the connections, the platform distinguishes measurements based on the connection's properties like geographical position or network provider. Hence, the system is able to manage measurement campaigns (henceforth campaigns), which group together experiments defined for connections according to well defined criteria. The administrator can define campaigns for connections established in a specific geographical area, for a set of municipalities, for a specific provider or service plan and so on. There exist two different selection criteria:

- *preferential*: connections are requested to meet some optional characteristics.
- *imperative*: connections are strictly requested to meet the defined characteristics with penalty of exclusion.

Measurement schedule. A measurement experiment may be more or less valuable according to the time of execution. For this reason, it is of the utmost importance to plan the experiments with a proper schedule, to allocate experiments in specific days of the week or temporal slices or during particular days like Christmas or Easter. The schedule implements a mechanism similar to the experiments rule in order to filter the experiments performed by each client. The management server communicates to the clients the list of experiments they should perform, encoded in XML format according to the specific connection currently in use. In order to avoid the overloading both of clients and measurement servers, the allocated measurements are limited such that temporal constraints are met: the sum of experiment durations should not exceed a predefined threshold. This mechanism, together with the slices allocated by the measurement servers, allows to limit the network resources used by the clients.

The list of measurements to be performed also contains a time reference:

- experiment duration;
- time to wait before the experiment;

- number of retries in case of failure;
- time to wait before a new list of measurements could be requested.

The experiment results are defined in terms of samples. For this reason, the list also contains a sample rate for each experiment. When the experiment provides a single aggregated value (e.g. mean, minimum, maximum, ...), such value is considered as a sample. Finally, for each experiment, the list contains possible outputs.

A specific syntax allows the definition of parameters and values. A null value indicates that the client must request the real value only when ready to start the experiment. In this way, the server is free to not specify all the parameters values during the creation of the list .

Measurement result. When all the experiments in the list are performed, the client sends the results to the management server, again in XML format. Each result consists of the set of samples collected during the measurements related to each output parameter. Since, in most cases, the list of results is not immediately communicated to the server, the list contains the time information about the end of each experiment. Such additional information could profitably help debugging operations.

Database design and implementation

We faced the problem of realizing a structure able to hold all the necessary information and, at the same time, to preserve the functional scheme defined above. The HoBBIT database is made of more than twenty tables and about ten functions and stored procedures. The core of this structure is the `Connections` table, since all the experiments are directly associated with its entries, from which them can be aggregated on several axes.

Design and implementation overview. From the elements gathered in the previous definitions, we build the conceptual model which organizes all the described elements and their associations. The HoBBIT database was built starting from that model, which is shown in Fig.4.2.

The most relevant entities have been highlighted in the diagram. Moreover, each entity belongs to a competence macro-area. The division in macro-areas is strictly conceptual

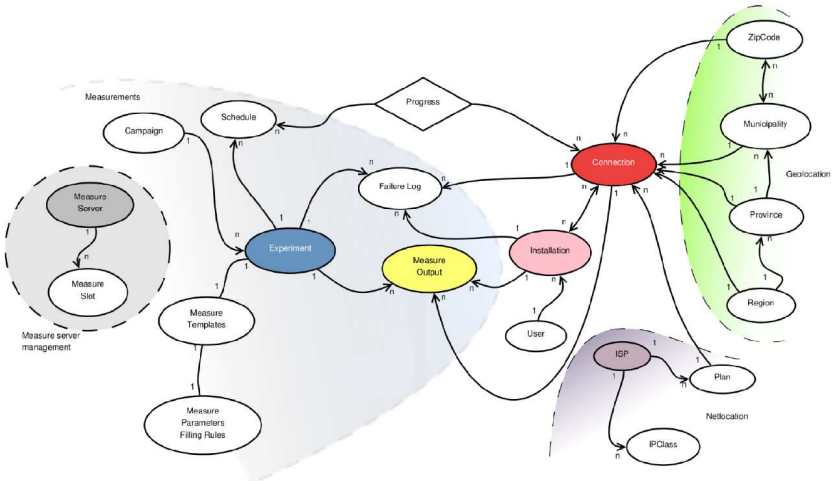


Figure 4.2: Conceptual design of the HoBBIT database.

and is aimed at explaining in which context the entities are operating and which information they hold. The entity representing the connection (`Connection`) emerges as the fundamental one for the project, as it describes, with the maximum allowed resolution, the link on which the measurements are performed. This entity relates with other entities belonging to the different macro-areas. By means of the geographic section, the mapping of the connection on the territory can be tracked. The Netlocation section determines the phone service contract, and indirectly the ISP, that is serving the connection. Finally, the Measuring section holds the experiments that are active and performed on the connection, and the results produced by them.

The entity comprising the information about the client application is named *Installation*. It is in a many-to-many relationship with Connection: this means that a connection can host many client applications and a client application can be hosted - not at the same time - on different connections. This kind of relationships implies that the application is not bound to performing statistics only on the first connection it happens to operate, but it can participate in different measurements. The Experiment entity is in charge of holding the measurement experiments defined by the system administrator. There is no direct relationship between this entity and Connection because the relationship is actually between the connection and one instance of an experiment at a given time. In order to better clarify this relationship, we introduce the concept of experiment scheduler (described in details in the previous paragraphs). As a matter of fact, an experiment is just the definition, according to some parameters, of a given measurement; on the other hand

the schedule is the instance of the experiment, in a time interval and with specified values.

In this way and by means of a many-to-many association, we relate a schedule to a connection, so that many experiments (schedules) can be executed on one connection, and one schedule can be associated with many connections.

The results of the experiments are stored in the entity `MeasureOutput`. In this entity only the information about the results (i.e. which experiment generated them, on which connection) is stored. A macro-area not strictly bound to the connection is the one related to the management of the measurement servers. The entities that belong to it are in charge of the management of the specific servers that participate to the experiments.

In the following we provide a brief description of HoBBIT 's database tables:

- `Connections`: holds all the information to uniquely identify connections, such as the MAC address of the gateway and an alias identifier (`AliasID`), which is used to associate among them different interfaces belonging to the same router, as illustrated in Fig. 4.3a and 4.3b.

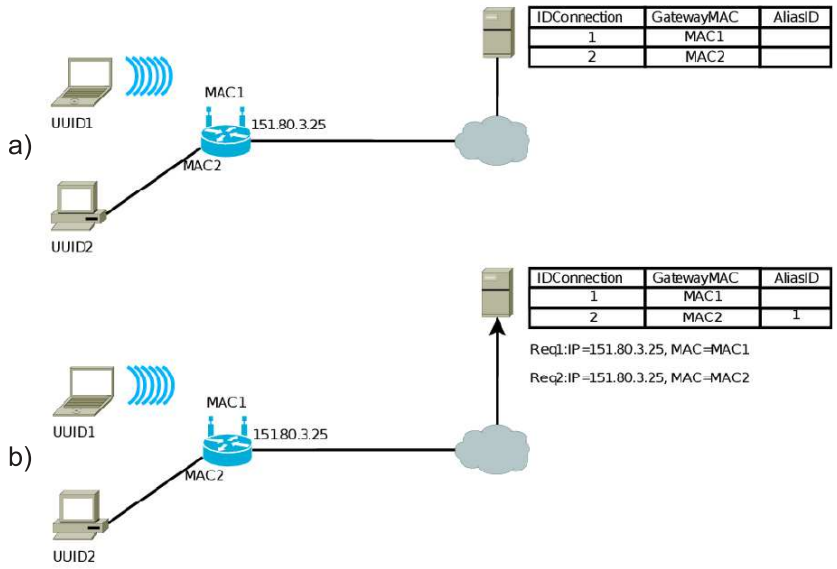


Figure 4.3: Example of AliasID operation.

- `Installations`: maintains all the information to identify a particular instance of the application client, which is uniquely identified by the `UUID` value.

- `Users`: holds the information about the users necessary identify them (i.e. using the email address) and to access the front-end server for visualizing their statistics.

- `ISPs`: maintains information about all the known ISPs.
- `Plans`: holds information about all the service plans provided by the ISPs.
- `IPClass`: maintains all the CIDR IP blocks associated to ISPs.
- `Campaigns`: holds all the experiment campaigns as defined by administrators, together with functions to dinamically associate connections with them.
- `Experiments`: maintains the experiments definition, including the complete set of input/output parameters and a `bash` script acting as a wrapper for underlying measurement tools.
- `Schedules`: holds all the schedules associated to the experiments (see Sec. 4.2).
- `Progresses`: maintains the progress of each connection with respect to each experiment.
- `MeasureOutputs`: holds all the meta-data associated with experiment results.
- `Samples`: maintains the samples returned by all the experiments.
- `FailureLogs`: holds a log of all experiments execution failures.
- `MeasureServers`: maintain basic information about the measurement servers (e.g. IP address).
- `MeasureSlices`: holds all the slices in which measurement servers resources are partitioned, in order to allow the execution of the scheduling algorithm.

The client/server protocol

Such protocol is mainly constituted of two fundamental parts depending on the location of the client in the network: the first one defines the case when the user contacts the server from a new station, thus using a connection different from those registered; the second one, instead, is related to the exchange of information between client and server when the client station is already registered and thus known to the server. This way, the user can use multiple connections and participate to different measurement campaigns. A special condition happens with the first exchange of information between client and

server. In this case, the installation is not active yet, being not registered on the server. A new installation gets registered through a specific query from the client to the server, which replies with a unique identifier confirming the successful registration. Since then, the client installation becomes part of the project and participates to its measurement campaigns.

The first client/server communication. The client/server protocol related to the first request from the client is described in Fig. 4.4. During the installation, the client sends an HTTP query to the server, specifying the name and version of its operating system and the MAC address of the network interface used to connect to the default gateway. In this query no installation identifier is specified, since this has not been assigned yet. The server generates a unique identifier (called UUID) associated with the client and sends it back to the client. The UUID will be used in all subsequent communications. At the same time the server looks for the client's MAC address in its connections database. If the MAC address is already present in the DB, the server will send all its geographical data associated with the connection, such as the City, Region, ZIP code, etc., plus the ISP associated with the connection and the service plan. In this special case the client will not need to estimate the properties of the connection, a relation between the connection and the new installation will be created, and the next step will be the registration of the user.

If the MAC address is not in the connection DB, a new connection will be created in the DB with the MAC address associated with it. The user will indicate the geographical data, and his commercial plan. The ISP instead is obtained server-side through a whois request. Afterwards, a relation between the connection and the installation is created. The client is then ready to execute its first measurements, and thus requests a list of the measurements that must be performed (this is provided in XML format).

When a new connection is created, a specific measurement campaign - named "first connection campaign" - is assigned. This is made of two experiments: an experiment for the estimation of throughput and an experiment to estimate delay, jitter, packet loss. The throughput estimation provides a measure of the capacity of the link, which allows us to evaluate the download rate of the link and, indirectly, the commercial plan of that connection.

When finished with the measurements requested by the server, the client sends the

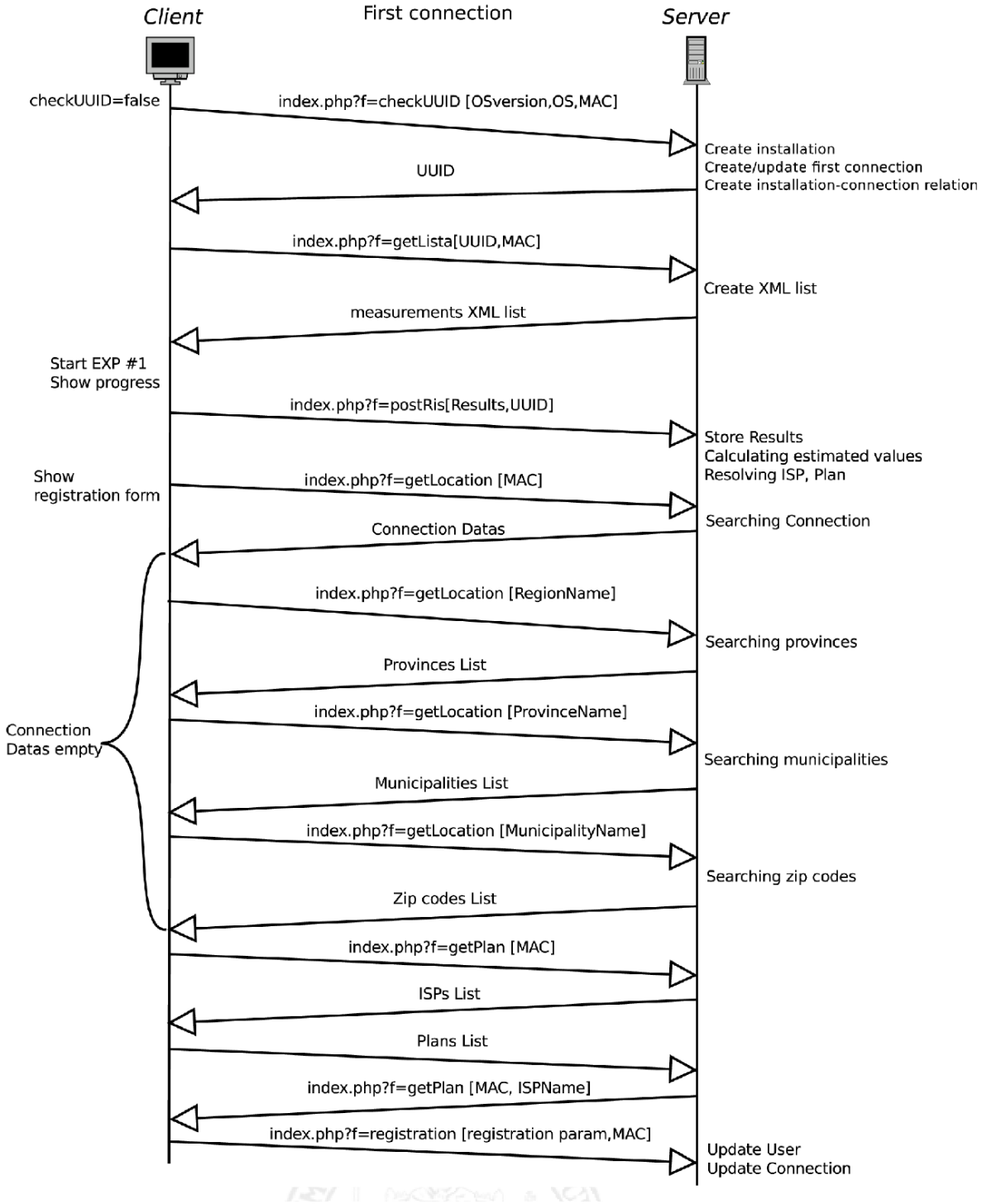


Figure 4.4: Client/Server protocol when detecting a new connection.

results of each of them in XML format to the server. The server processes the results and performs an estimate using the collected samples (it calculates the 95th percentile for

downstream throughput and the 5th percentile for the upstream throughput). Afterwards, the ISP is identified and the server is ready to finalize the registration of the new client installation. Besides the geographical data, the user can optionally add Zone and Place information. The possible values for the Zone are: industrial area, downtown, historical area, island, periphery, hills, seaside, mountains, rural area. The user in the beginning is identified through his email address. After the registration procedure, the server will update the information associated with the connection and will insert or update the user's data.

Following client/server communications. The subsequent communications will approximately follow the same scheme of the first connection. Before any request, the client will send its UUID and the MAC address of its gateway to the server. The server will first verify the correctness and the presence in the DB of the UUID, and then will check for the MAC address:

- If there is a relation between the installation and the connection, then it means that the client is behind a gateway from which it previously connected to the server. In this case, the next step will be the request of a list of measurements.
- If the relation does not exist, it means that the client is using a new connection and therefore it is necessary to perform a new registration as described earlier.

The described mechanisms provides more flexibility to the application in the case it runs on mobile devices as notebooks, smartphones, etc. This way, a user changing position and connection can participate to multiple measurement campaigns and can actively contribute to reaching the goals of the project.

Other features

Client registration process. As explained earlier, before issuing any request, the client's installation is verified by the server through the UUID. This verification is used to verify that the client is still active and to detect situations in which the client moves to a different connection.

Let us see in detail the situation in which the user, after installing the client, connects to the server. In this case, the UUID cannot be sent, because the client does not have one.

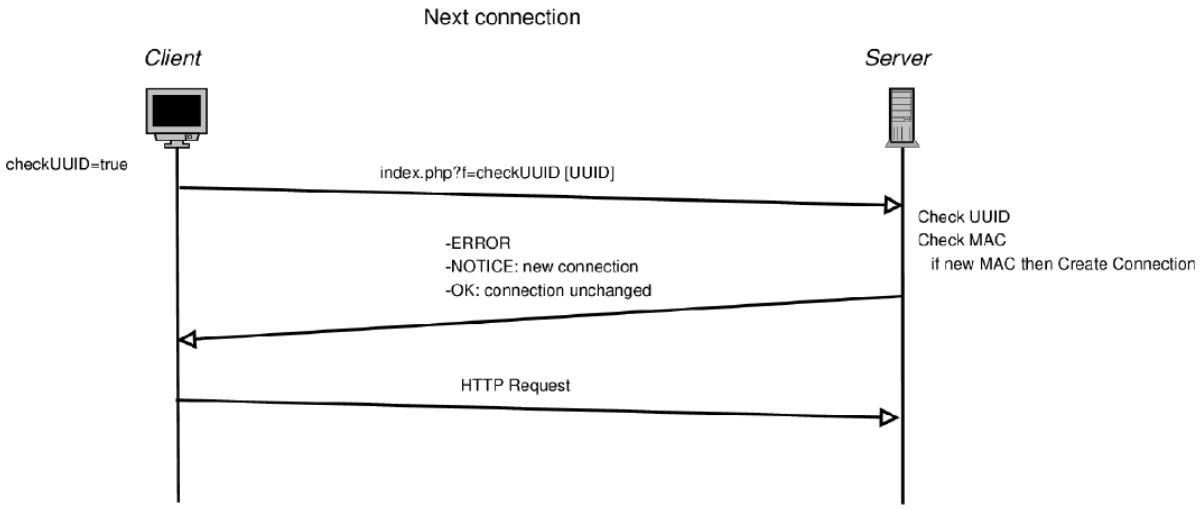


Figure 4.5: Client/Server protocol when detecting an existing connection.

The server recognizes the absence of an UUID and interprets the situation as a request of an identifier. A new client installation is registered and the new UUID is sent to the client. The client sends the MAC address of its default gateway, which identifies a connection. The server verifies the presence of the MAC address in the DB. If present, it creates a relation between the existing connection and the new installation. If the connection is not in the DB, a new connection is inserted in the DB with the MAC address passed by the client. After that, the default "first connection campaign" is assigned to the connection, after which the client is ready to perform the measurements requested by the server. It is worth noting that if the connection was already present, the client does not have to perform the "first connection campaign" and it can directly proceed to the user registration.

Let us discuss now the case in which the client is already registered. The client requests the verification of the UUID and of the connection to the server. If the UUID is not valid anymore (because the installation has been deleted from the server, or because the UUID is corrupted) the server will send an error message to the client, which will proceed for a new registration. If the UUID is valid, the server will search the relation between the installation and the MAC address sent. This step is necessary in order to verify that the user is using a connection already registered so that it is possible to proceed with the measurements. If the relation does not exist it will be created, inserting the connection if not in the database and notifying the client of the changes.

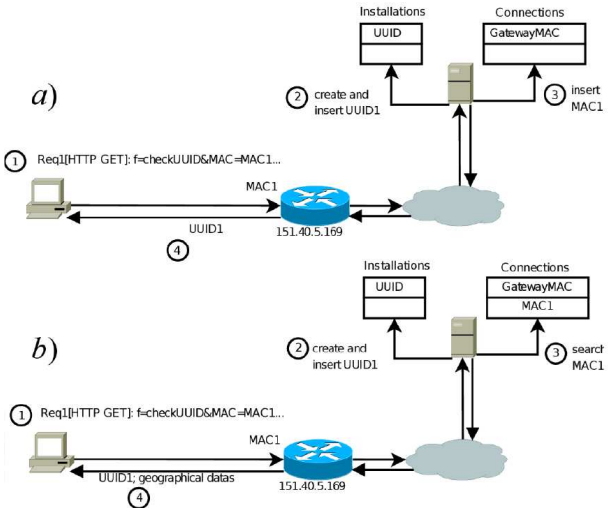


Figure 4.6: UUID requests from HoBBIT clients. (a) The client is not registered yet. (b) The client is already registered.

Scheduling algorithm for measurement resources. A set of slices are assigned to each measurement server, depending on the available bandwidth. The slices are stored in the `MeasureSlices` table, which is composed of the following fields:

- `IDSlice`: unique ID of the slice;
- `IDServer`: ID of the measurement server which the slice belongs to;
- `FREE_TS`: the time, in Unix Epoch format, when the slice will be free again.

The concept of slice is strictly related to the network capacity of the measurement server. Each of these servers is connected to the network through a 100 *Mbps* Ethernet link; we partitioned such capacity in 180 slices of 512 *kbps* each, reserving some free periods to avoid bottlenecks and interference between measurements.

When an experiment requires a certain number of slices, those available are verified and reserved by setting the `FRESS_TS` field to the sum of the time in which the slice has been requested, the duration of the experiment and a guard time, to avoid overlapping with other experiments. An example is reported in the following.

Let us refer to Fig.4.7 and suppose that a request to perform the experiment #1 is issued at time 1000 and that the all the slices of the measurement server are free. This experiment requires a bandwidth of 1024 *Kbps* (i.e. two slices), and it has a duration of 15 seconds. The measurement can take place and the slices required are reserved for the duration of the experiment; therefore the first slices will have a value equal to:

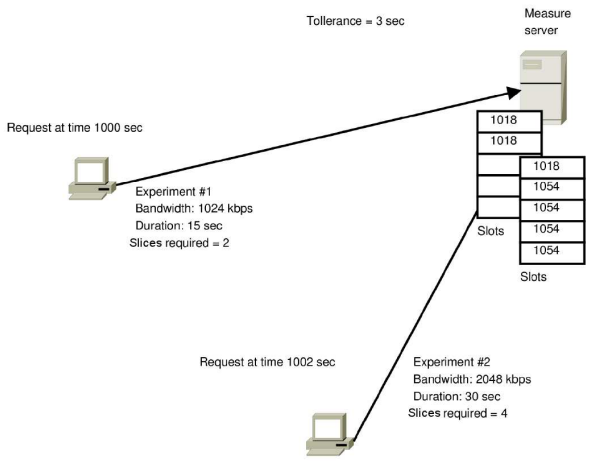


Figure 4.7: Example of scheduling algorithm in action.

$$FREE\_TS = Duration + Tollerance \tag{4.1}$$

Afterwards, at time 1002, the server receives another measurement request for the experiment #2. It requires a bandwidth of 2048 *Kbps* (i.e. 4 slices) and a duration of 30 seconds. As there are only 3 slices available, it is necessary to postpone the execution of the experiment up to when a new slice is free. The waiting time is given by the difference between the time in which the slice will be free and time in which the request is received:

$$Wait = time_{release} - time_{now} + tollerance = 1018 - 1002 + 3 = 19s \tag{4.2}$$

This waiting time is sent to the client which requested the measurement, so that it can postpone the experiment. At this point, it is necessary to reserve all the slices adding to the time in which the request is received the waiting time, the duration of the experiment and the guard time:

$$FREE\_TS = time_{now} + wait + duration = 1002 + 19 + 30 + 3 = 1054 \tag{4.3}$$

Measurement list creation and delivery The exchange of information between client and server happens through HTTP messages in XML format. When the client asks for a list of measurements, the server answers with an XML list of experiments. After the UUID and the MAC address have been verified, the connection is identified. Then,

the experiments associated with the connection are chosen using the following criteria:

- it is possible to execute them at the current time;
- it is possible to execute them in the current day of the week or day of the month;
- the time at which they have been assigned is smaller than the current time plus a tolerance value;
- they have a number of remaining repetitions larger than zero.

If none of the experiments verifies the above conditions, the client will receive an empty list of measurements, otherwise the parameters will be parsed. Each parameter is separated by the character ";". A typical string of parameters is:

$$value1; value2; \S foo(p1, p2); value3; \S\S bar(p1) \tag{4.4}$$

In this case the function "foo" is called when the list is created, whereas the function "bar" is called upon client's request; up to then the value of the corresponding parameter will be zero. For each experiment a slice of a measurement server is assigned. The slice will contain the time interval in which it will become free and is expressed as a timestamp in Unix epoch format, to which the duration (in seconds) of the experiment is summed.

Not all measurement parameters are statically defined by the administrator. Some of them are variables that are dynamically set by a request from the client. An example is assigning a slice to a measurement server for a particularly invasive experiment. In this case, when the list is created, the value of such parameters will be empty and it will be automatically filled upon client's request. The client specifies the values of the parameters when the request is made. The server, through predefined functions, will calculate the proper values and will send them to the client.

Measurement results parsing The results of the measurements performed by the client are sent to the server in XML format. The server processes the results and stores the samples of each measurement and some information on the results, among which:

- the IP address of the measurement server;
- the IP address of the measuring client;

- the measurement parameters;
- the UUID associated with the client's installation.

Such results, together with the samples of each measurement, are stored and used for statistics and aggregations.

Plan detection After sending the results, for each new connection it is necessary to associate the corresponding service plan and ISP. The CIDR IP class to which the client's IP belongs is searched for, in order to identify the corresponding ISP. If the corresponding IP class is not found, a `whois` query is issued. Once the ISP has been identified, using the estimated throughput of the connection, the service plan of the connection is identified and associated with it. If no service plan is identified, a default one is created using the name of the ISP and the estimated download rate. For example, if we identofy Infostrada as the provider, and a download rate of 2 $Mbps$ and upload rate of 400 $kbps$ is measured, the service plan will be named "Infostrada 2M/400k" and will be associated to the connection.

Connection registration process During the registration, the client does not know if the connection is already registered on the server. For this reason, the client sends an HTTP query to the server indicating the MAC address of its default gateway. The server looks into the database for the connection and sends the geographical information that identify the connection, if present. The geographical information includes: region, province, city, postcode, and characteristics of geographical area (e.g. center, periphery, ...).

If not found, the connection needs to be registered and the server sends an empty string to the client. The client is then ready to register the connection.

If the connection needs to be registered then the client and server will exchange the geographical information. Optionally, the user can suggest the specific geographical area in which the connection resides. After the geographical information, the client must obtain the information related to the ISP and the commercial plan. The client sends a request to the server, which sends a list of available ISPs. Among these, the server includes the ISP obtained through the `whois` query. After selecting the ISP, the client sends to the server another request: a list of commercial plans associated with the selected ISP. The server sends the list of all commercial plans and selects the one that has been identified through

the throughput estimates as preferred. At this point, the connection is registered with the parameters selected by the user. The server inserts or updates the connection, the user and the installation in the DB. After the registration, a set of measurement campaigns is associated with the registration.

Measurement campaign assignment After the registration of a new connection and after the creation of a new measurement campaign, a list of experiments, related to one or more campaigns, is assigned to all the connections verifying specific criteria. When a new connection is added, all the campaigns that can be associated with it are selected. The association happens through a list of functions, related to the campaign, which allows to discriminate the connections that satisfy the characteristics described by the input parameters of the function. Such functions return a list of connections candidate for the association with a campaign. In the case a new connection is registered, the functions accept as input also the connection identifier and return the identifier if the connection is a possible candidate for the measurement campaign. If the connection satisfies the criteria of all the functions, then it is associated with the corresponding measurement campaign.

Automatic software update To implement the version control, all the files used by the client are kept under version control on a Subversion repository hosted by the Management Server. Every time a client requests a version check, by passing its current version to the related PHP script, the latest available version is obtained executing the following command:

```
svn status -u ../dl \| awk \'{ print $4 }\'
```

Then, if the latest version number is larger than the current version number, the list of files modified between the two versions is obtained by executing the following command:

```
svn diff --summarize -r1 svn://127.0.0.1/
```

Such list is then encoded in XML format and sent back to the client.

4.1.2 The measurement client

In this section we provide a detailed description of the design of the measurement client, by giving also some insights on the choices adopted to cope with their mobility and to

maintain a good flexibility for the execution of the experiments.

Requirements analysis

The HoBBIT client has been designed taking into account the guidelines described in the following. On one hand, the client should be able to properly associate experiment results with a geographical location and with the correct provider and service plan. We are interested to obtain location information with the maximum detail of the postcode. On the other hand, it should also be able to identify the user and the connections on which it performs the experiments, thus allowing to easily point to the related results on the front-end Web interface, by optionally requesting an authentication procedure.

When a client is installed for the first time, the registration procedure should provide three phases:

- Estimation of the parameters characterizing the connection to be able to detect the service plan;
- Collection of the geographical location of the connection and of the email address of the user;
- Registration of all the information collected on the management server.

The three steps above should be repeated in the case a client is detected on a connection which was never seen before by the management server. After being properly registered, the client has to periodically request instructions to the management server about the experiments to execute. The results obtained should be sent to the management server as soon as possible. The client should also provide an automatic update procedure, which would allow us to remotely solve bugs or add new measurement tools or features without requiring user intervention.

In general, in order to reach a wide deployment, the client should be available for the most important operating systems (e.g. Microsoft Windows, Apple OSX, Linux) and should satisfy the following properties:

- *Simplicity*: the interfaces provided to interact with the client should be clear, intuitive and essential;

- *Non intrusiveness*: the client should be as quite as possible to avoid the user to be annoyed by its presence. This is really important because we point to obtain long term analysis of performance and implies that the impact of the experiments should not be perceived by the user.
- *Flexibility*: the client should provide the support for any underlying tool and should be able to properly manage their input/output parameters as requested by the management server.
- *Respect of users' privacy*: it should collect only the information strictly necessary to reach the goal of the project, thus it should just keep the geographical location of the connection with a coarse-grained precision (e.g. the postcode).

Client design

We designed the HoBBIT client by following the guidelines described above. We adopted an object oriented approach supported by the UML (Unified Modeling Language) language. In the following paragraphs we describe all the objects composing the client application.

Application overview. As shown in the class diagram in Fig. 4.8, the HoBBIT client is made of eight classes:

- `Connection`: manages all the communications with the management server;
- `LogicUnit`: takes all the decisions on when to perform any action;
- `List`: manages the execution of a list of experiments;
- `Experiment`: manages the execution of a single experiment;
- `Update`: manages the automatic update procedure;
- `GuiGetInfo`, `GuiWelcome`, and `Icon`: manage the graphical interface elements.

We give more insights on each class in the following paragraphs.

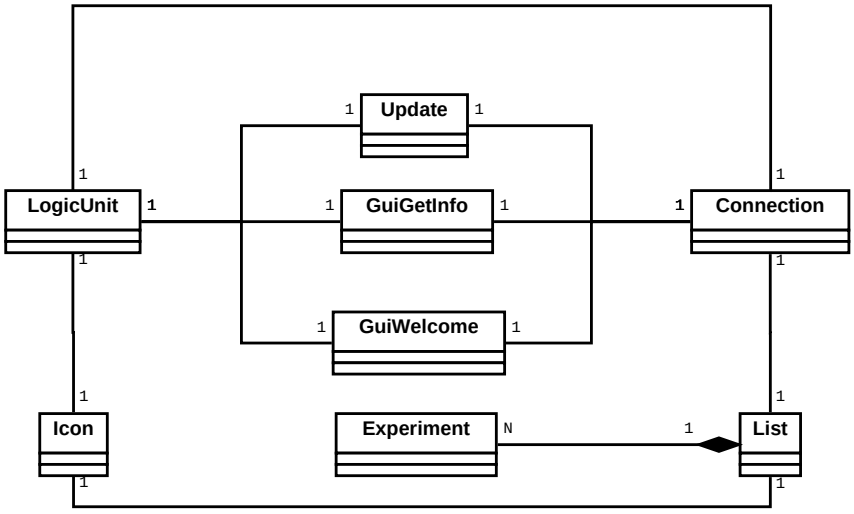
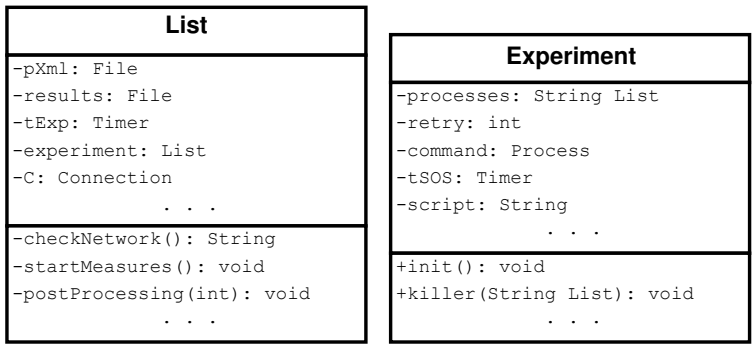


Figure 4.8: HoBBIT client class diagram.

The `List` class. The `List` class is responsible for managing lists of experiments. We describe its behavior referring to Fig. 4.9(a). Using a timer (`tExp`), it periodically requests the list to the management server through the `Connection` object, which returns it as an XML file (`pXML`). It parses such file to extract the list (`experiments`). It then checks for the status of the network interface (`checkNetwork()`) and in turn, if the current traffic is above the configured threshold, starts the execution of each experiment by calling the `Experiment` class. When the experiment is finished, it collects the results (`postProcessing()`) and writes them to file (`results`). When all the experiments contained in the list are finished, it sends the result file to the management server through the `Connection` object. In order to avoid to loose the results of the experiments if the client is



(a) (b)

Figure 4.9: The `List` and `Experiment` classes.

interrupted during the execution of a list, it scans the temporary folder for partially composed XML files. Any of those files is properly completed and sent to the management server before requesting a new list.

The `Experiment` class. The `Experiment` class is responsible for executing a single experiment. We describe it referring to Fig. 4.9(b). When instantiated (`init()`), it receives from `List` all the information necessary to execute the experiment: a set of input and output parameters and a script (`script`) which utilizes such parameters to run the underlying tools. It adds to the script the variables corresponding to the input/output parameters and stores it as a temporary file. It then executes the script (`command`) and keeps under control all the processes generated by it (`processes`). If the execution lasts more than twice the expected duration (`tSOS`), the experiment is interrupted together with all its processes (`killer()`) and it is repeated up to a maximum number of times (`retry`) before giving up.

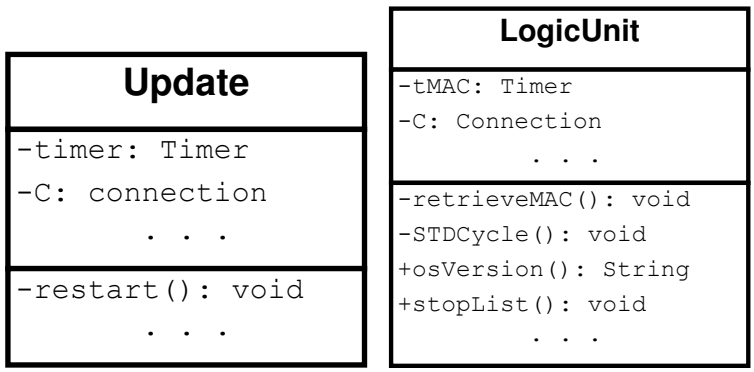


Figure 4.10: The `Update` and `LogicUnit` classes.

The `Update` class. The `Update` class is responsible for managing the software updates. We describe it referring to Fig. 4.10(a). Using a timer (`timer`), it periodically contacts the management server, through the `Connection` class, to verify if a new version of the software is available. If so, only the files modified since the current version are downloaded to a temporary directory and then the client, after notifying the user, is automatically rebooted (`restart()`).

The `LogicUnit` class. The `LogicUnit` class is responsible for managing and synchronizing the interactions among all the other classes. For instance, it avoids the update procedure to be initiated if an experiment is being executed. We describe it referring to Fig. 4.10(b). When the HoBBIT client is executed, it creates a `LogicUnit` object which is then responsible for creating most of the other objects. Using a timer (`tMAC`), it periodically retrieves from the local ARP table the MAC address of the default gateway (`retrieveMAC()`), which is sent to the management server, through `Connection`, to check if the access network has changed. Moreover, it manages all the periodic activities related to the execution of the experiments (`STDCycle()`) and can request the their interruption (`stopList()`). It is also responsible for detecting the version of the operating system (`osVersion()`).

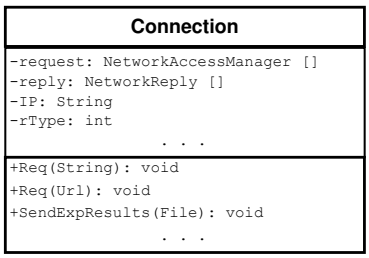


Figure 4.11: The `Connection` class.

The `Connection` class. The `Connection` class is responsible for managing all the network communications, hiding the their complexity to the other objects. We describe it referring to Fig. 4.11. By using `NetworkAccessManager` objects it is able to send requests (`Req()`) to a server (`IP`) and to asynchronously receive the reply using the associated `NetworkReply` object. It is also capable to send and receive files using the same mechanism. Particularly, it is responsible to send the results obtained by the experiments to the management server (`SendExpResults()`) and to request it specific information (`rType`).

The `GuiWelcome` class. The `GuiWelcome` class is responsible for managing the first graphical interface shown to the user after installing the software, as shown in Fig. 4.12. We describe it referring also to Fig. 4.13(a). It first asks the management server to check if the current connection is already known (`checkConnection()`). If not, it shows (`gui_show()`) its graphical interface (`Welcome`) to inform the user that an estimation of

Figure 4.12: Screenshot of the *GuiWelcome* graphical interface.

the connection parameters is required and he should stop all the applications which use the network before continuing. After the user clicks on "OK", it waits for the network to be effectively unloaded (`checkNetwork()`) and starts a special campaign of experiments (see Sec.4.2.1). The progress of the campaign is shown through the progress bar, giving also a description the experiment currently in progress (`setText()`, `description`). At the end of the process, or if the connection is already known but lacks of some information, it passes the control to the `GuiGetInfo` object.

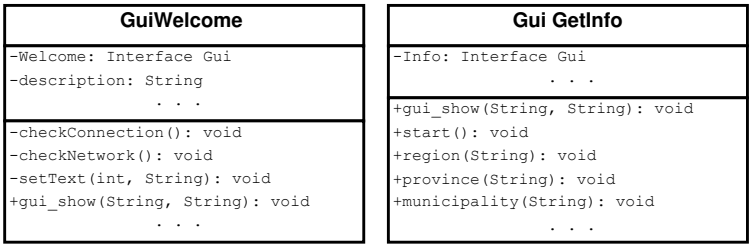
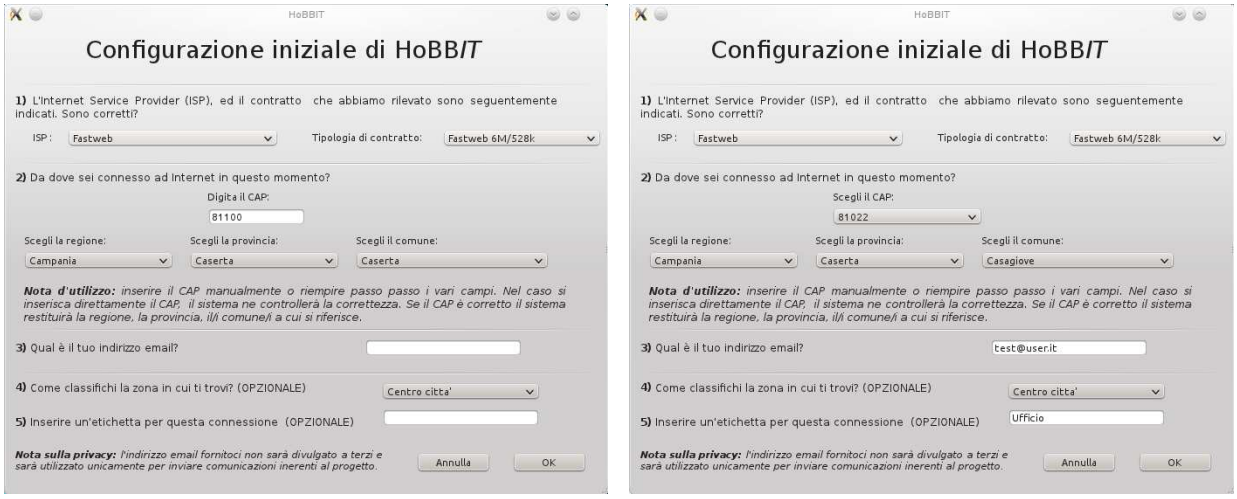


Figure 4.13: The `GuiWelcome` and `GuiGetInfo` classes.

The `GuiGetInfo` class. The `GuiGetInfo` class is responsible for managing the graphical interface shown to the user when requesting to it information about his geographical location and to complete the registration process for the connection. We describe it referring to Fig. 4.13(b) and 4.14. It shows the ISP and service plan information (`gui_show()`), as inferred from the results of the special campaign of experiments performed, and allows the user to correct such result. The interface (`Info`) can appear in two different forms depend-



(a) When the user directly insert the postcode (b) When the user selects step-by-step the location

Figure 4.14: Screenshot of the `GuiGetInfo` graphical interface.

ing on the user behavior. If the user directly inserts the postcode (see Fig. 4.14(a)), the interface, automatically populates the other geolocation fields after requesting the necessary information to the management server (`region()`, `province()`, `municipality()`). Otherwise, if the user selects step-by-step region, province, and municipality (see Fig. 4.14(b)), it transforms the postcode text field into a select box and populates it with all the possible postcodes associated with the selected municipality. Once the user has filled in all the mandatory fields and has clicked on "OK", all the information is sent to the management server to complete the registration of the connection, which starts the normal operations (`start()`).

The `Icon` class. The `Icon` class is responsible for managing the icon placed into the tray bar, as shown in Fig. 4.15. The icon can appear in three different status: *static*, which indicates that the application is active, but no experiment is being performed; *rotating*, which points out that an experiment is in progress; *grayed*, which means that the application has been disabled by the user. By using the contextual menu the user can perform different actions. If he wants to avoid to be annoyed by the experiments while using another application (e.g. video streaming), he can disable the application for a time interval up to 30 minutes or until the next boot. Since automatically detecting a change of service plan or ISP may be challenging, the user can notify such event explicitly by modifying such information on the `GuiGetInfo` form. He can access the front-end Web

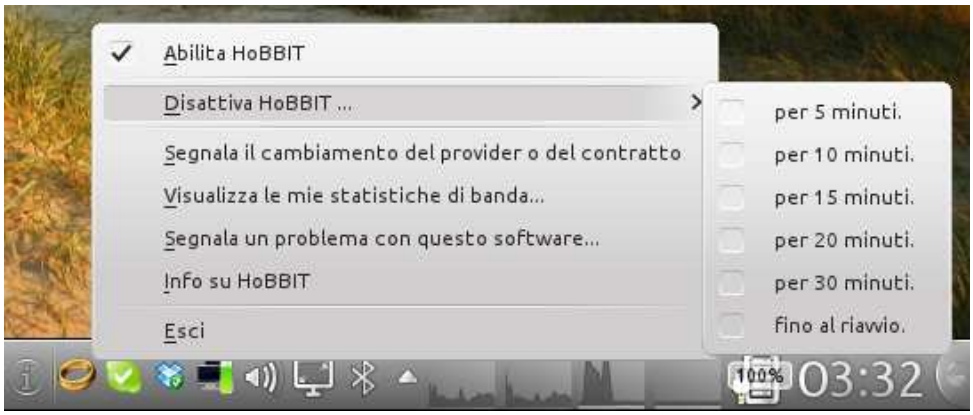


Figure 4.15: Screenshot of the `Icon` contextual menu.

interface to visualize the statistics associated to his connections. Moreover, the user can easily send a report to the developer team in case of problems. The icon is also used to notify the user in case a new version of the software is detected, but no intervention is required by the user to perform the update procedure.

Implementation choices

To implement the HoBBIT client we chose the Qt framework [108], which is one of the most spread multi-platform libraries. Since we want to execute experiments by exploiting any underlying measurement tool, we had to provide the client with a set of utilities. We chose `bash` as the scripting language for implementing the wrapper for each tool, which is responsible for passing input parameters, retrieving outputs and transforming them in XML format. We use `gawk` as interpreter to parse tools output on the fly. We use `wget` to implement HTTP based communications between the client and a measurement server in order to perform specific experiments.

All the above mentioned tools are not natively multi-platform. Since they are all supported on Unix/Linux operating systems, they are also available for Apple OSX. In order to support also Microsoft operating systems, we had to include also the cygwin libraries [109], which include a ported version of all of them.

4.1.3 Measurement servers

HoBBIT platform currently includes two servers dedicated to measurements hosted at the University of Napoli. They are equipped with Gigabit Ethernet interfaces and their

access to the Internet has a capacity of about 200 Mbps. With respect to BISmark , in HoBBIT the resources are managed differently using a more sophisticated mechanism.

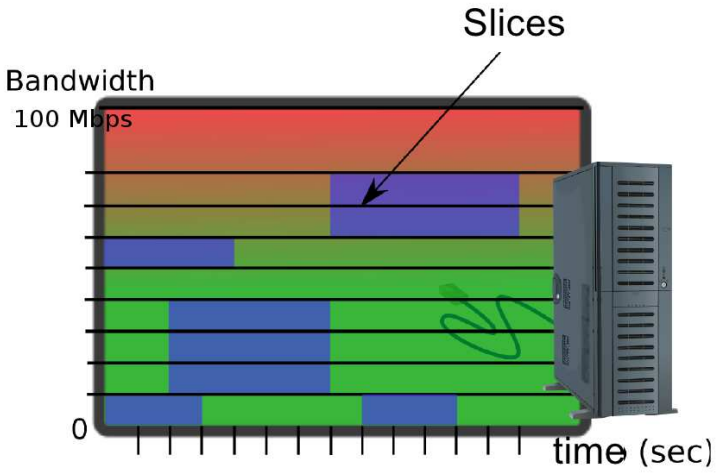
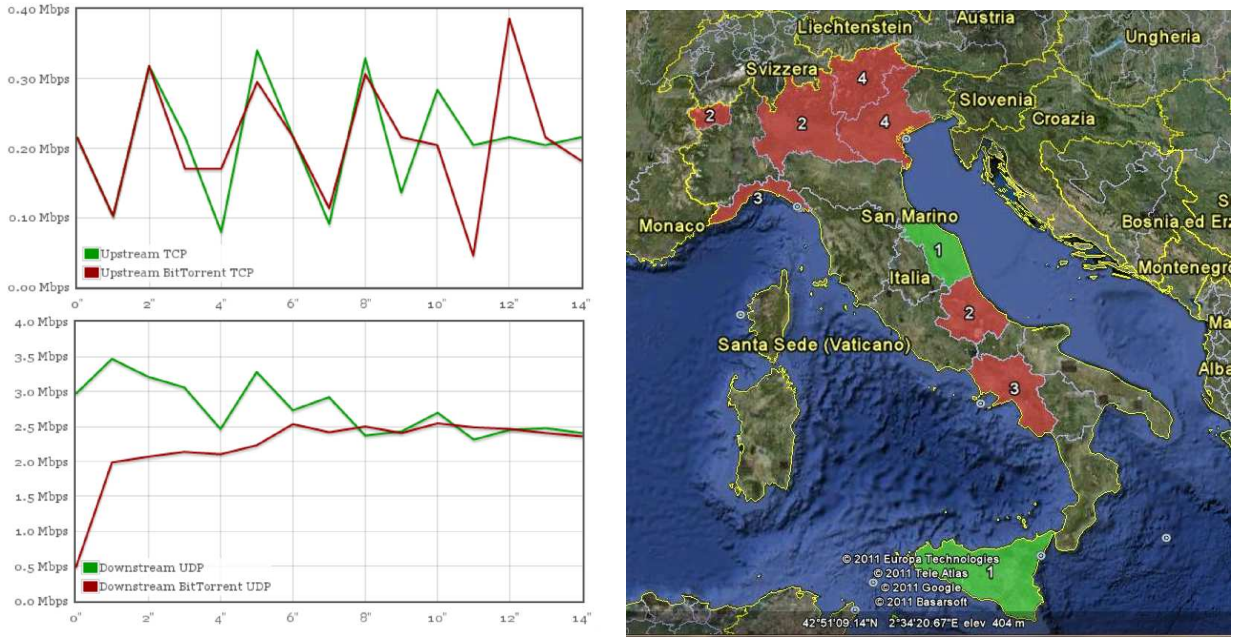


Figure 4.16: Slicing of measurement servers network resources.

As shown in Fig. 4.16 the capacity of each server is partitioned in *slices* of 512 *Kbps*, resulting in 200 slices available on each server. To be conservative, we consider only 180 slices for each server. The scheduling algorithm implemented by the management server allows to assign a measurement to a specific server only if enough slices are available on the same server. If the measurement has a duration of 30 seconds and requires 5 Mbps of bandwidth, then 10 slices will be assigned to it for 30 seconds. When considering a slice assigned for a certain time interval, we refer to it as *timeslot*. Each slice is associated to an independent `FREE_TS` which is managed as described in Sec. 4.1). By implementing this mechanism in this way, we gained a lot of scalability , because we allow the overlap of invasive measurements until the capacity of the measurement server is close to saturation.

4.1.4 Front-end and map servers

The HoBBIT platform also includes a front-end Web server which allows users to visualize the statistics collected by analyzing the experiments conducted on their connections. Currently users can show plots about each experiment result. For instance, Fig. 4.17(a) shows two plots related to single upstream TCP and downstream UDP throughput experiments for the same connection. In addition to the front-end server, the HoBBIT platform also includes a map server which is based on GeoServer [110] and allows to visualize statistics aggregated by geographical areas on fancy maps, as shown in Fig. 4.17(b).



(a) Plots of upload and download experiments result. (b) Geographical distribution of some HoBBIT clients.

Figure 4.17: Screenshot of data visualizations on the front-end interface.

4.2 Measurement campaigns

As explained in Sec.4.1.1 HoBBIT organizes experiments into campaigns which can be dynamically associated with client installations. In the following sections we describe the campaigns currently operating on a deployment of about a hundred clients in Italy.

4.2.1 Connection parameters estimation (CPE) campaign

When a client operates on a specific connection which is seen by the management server for the first time, this special campaign is executed to estimate its performance parameters, reported in Tab. 4.1. All the metrics are computed end-to-end by using the UDP transport protocol.

Table 4.1: Experiments part of the CPE campaign.

ID	Measured metrics	Required bandwidth (kbps)	Duration (sec)	Measurement class
#1	Latency, Jitter & Packet loss	512	30	LIGHT
#2	Upstream throughput	2000	15	INVASIVE
#3	Downstream throughput	20000	15	INVASIVE

The results obtained with this campaign are utilized by the management server to infer the service plan associated with the connection, whose ISP is detected by comparing the result obtained by a *whois* protocol query and the entries in the `ISPs` table. As reported in Tab. 4.2 it is executed only once for each connection at any time, unless a service plan or provider change is automatically detected or solicited by the user. All the experiments are implemented using the D-ITG tool [91].

Table 4.2: Schedule defined for the CPE experiments.

DoW	DoM	MoY	TimeStart	TimeStop	Repetition	FilterFuncion
			00:00:00	24:00:00	1	

Experiment #1: RTT, jitter and packet loss

As reported in Tab. 4.3, the first experiment defined for the CPE campaign is classified as LIGHT, requires just one slice to be executed (512 Kbps), and can be repeated only after 15 minutes. Its execution produces three outputs with sampling rate of one second: round-trip time, jitter, and packet loss.

Table 4.3: Outputs returned by the experiment #1.

Outputs	Class	Retry	SamplingRate	RequiredBandwidth
rtt (ms) jitter (ms) pkloss (%)	LIGHT, 900	3	1000	512

Table 4.4: Parameters defined for the experiment #1.

Name	Value	Param	IsFunction	Post	Description
ITGSend	getOSBased	{$UUID, bin bin bin}	true	false	OS dependent ITGSend path
ITGDec	getOSBased	{$UUID, bin bin bin}	true	false	OS dependent ITGDec path
gawk	getOSBased	{$UUID, sys bin sys}	true	false	OS dependent gawk path
server	getServerIP	{$IDExperiment}	false	true	Measurement server IP address
port	3000		false	false	Measurement service port
proto	UDP		false	false	Measurement transport protocol
pps	10		false	false	Measurement packet rate
psize	56		false	false	Measurement packet size
snd_log_file	/tmp/snd_log_file		false	false	Log file path

Tab. 4.4 reports all the optional input parameters of the experiment. Particularly, it shows that the experiment is configured to generate 10 UDP packets per second having a size of 56 bytes. It can be noticed that the only parameter requested just before the execution of the experiment (see Post column) is the server IP address, which allows to properly execute the scheduling algorithm on the management server.

```
# Execute experiment
$ITGSend -a $server -T $proto -C $pps -c $psize -t ${duration}000 -m rttm \
 -Sdp $port -rp 0 -l $snd_log_file

# Result XML generation
for out in ${!output[@]}; do
 # Result header
 echo "  <measure expid='$expid' output='$out' scheduleid='$schedule' dstip='$server' timestamp='$now'>"
 echo "    <parameters value='$params'/>"
 echo "    <samples sampling_rate='$srate'>"

 # Generate and parse samples
 case $out in
 1) option="-d" ;;
 2) option="-j" ;;
 3) option="-p" ;;
 esac

 $ITGDec $snd_log_file $option $srate $stdout | $gawk '(NR > 1) { print "      <sample>" $2 "</sample>" }'

 # Result footer
 echo "    </samples>"
 echo "  </measure>"
done >> $results_file
```

Figure 4.18: Script defined for the experiment #1.

Table 4.5: Outputs returned by the experiments #2 and #3.

Outputs	Class	Retry	SamplingRate	RequiredBandwidth
upload throughput (kbps)	INVASIVE, 1800	3	1000	2048
download throughput (kbps)	INVASIVE, 1800	3	1000	20480

Input and output parameters are used in the context of the script reported in Fig. 4.18, which is defined inside the `Script` column of the `Experiment` tuple. It takes care of the complete measurement process and returns the outputs already in XML format, by parsing the output of the `ITGDec` tool on-the-fly with the `gawk` interpreter.

Experiment #2 and #3: upload and download throughput

The definition of the experiments to estimate upload and download throughput is very similar to the previous one. As reported in Tab. 4.5, the main difference consist in their classification as INVASIVE, which allows to repeat them only every half hour. This is motivated by the fact that the upload and download throughput measurements respectively require 4 and 40 slices to be executed.

The input parameters required by the two experiments are reported in Tab. 4.6 and 4.7. The upload throughput experiment is performed generating UDP packets at a constant bitrate of 2 Mbps, by using packets of 1472 bytes. The download throughput experiment is performed generating UDP packets at a constant bitrate of 20 Mbps, by using again packets of 1472 bytes. In both experiments, as also reported in Fig. 4.19 and 4.20, the measurements are implemented by using also the `wget` tool. In the first case it is utilized to request the result of the experiment to the measurement server, as computed on the receiver side. In the second case, instead, it allows to initiate the traffic generation from the server side. In both cases the metrics are evaluated on the receiver side.

Table 4.6: Parameters defined for the experiment #2.

Name	Value	Param	IsFunction	Post	Description
ITGSend	getOSBased	{$UUID, bin bin bin}	true	false	OS dependent ITGSend path
gawk	getOSBased	{$UUID, sys bin sys}	true	false	OS dependent gawk path
wget	getOSBased	{$UUID, sys bin sys}	true	false	OS dependent wget path
server	getServerIP	{$IDExperiment}	false	true	Measurement server IP address
port	3000		false	false	Measurement service port
proto	UDP		false	false	Measurement transport protocol
kbps	2000		false	false	Measurement bitrate
mtu	1472		false	false	Measurement max packet size

```
# Compute parameters
remote_log="${uuid}_$now"
pps=$(( ( ${kbps} * 125 ) / ${mtu} ))

# Execute experiment
$ITGSend -a $server -T $proto -C $pps -c $mtu -t ${duration}000 \
         -Sdp $port -rp 0 -x /tmp/ITGRecv/$remote_log

# Result XML generation
(
 # Result header
 echo "  <measure expid='$expid' output='1' scheduleid='$schedule' dstip='$server' timestamp='$now'>"
 echo "    <parameters value='$params'/>"
 echo "    <samples sampling_rate='${srate}'>"

 # Generate and parse samples
 $wget -O - -q "http://$server/getlog.php?log=$remote_log&srate=$srate" | \
       $gawk '(NR > 1) { print "      <sample>" $2 "</sample>" }'

 # Result footer
 echo "    </samples>"
 echo "  </measure>"
) >> $results_file
```

Figure 4.19: Script defined for the experiment #2.

```
# Compute parameters
pps=$(( ( ${kbps} * 125 ) / ${mtu} ))

# Launch remote ITGSend in passive mode
$wget -O $trash -q \-\-post-data="-H -T $proto -C $pps -c $mtu -t ${duration}000 -rp 0 \
      -Ssp $port" "http://$server/startitgsend.php"

sleep 2

$ITGRecv -H $server -l $rcv_log_file -Sp $port

# Result XML generation
(
 # Result header
 echo "  <measure expid='$expid' output='1' scheduleid='$schedule' dstip='$server' timestamp='$now'>"
 echo "    <parameters value='$params'/>"
 echo "    <samples sampling_rate='${srate}'>"

 # Generate and parse samples
 $ITGDec $rcv_log_file -b $srate /dev/stdout | $gawk '(NR > 1) { print "     <sample>" $2 "</sample>" }'

 # Result footer
 echo "    </samples>"
 echo "  </measure>"
) >> $results_file
```

Figure 4.20: Script defined for the experiment #3.

Table 4.7: Parameters defined for the experiment #3.

Name	Value	Param	IsFunction	Post	Description
ITGSend	getOSBased	{$UUID, bin bin bin}	true	false	OS dependent ITGSend path
ITGDec	getOSBased	{$UUID, bin bin bin}	true	false	OS dependent ITGDec path
ITGRecv	getOSBased	{$UUID, bin bin bin}	true	false	OS dependent ITGRecv path
gawk	getOSBased	{$UUID, sys bin sys}	true	false	OS dependent gawk path
wget	getOSBased	{$UUID, sys bin sys}	true	false	OS dependent wget path
server	getServerIP	{$IDExperiment}	false	true	Measurement server IP address
port	3000		false	false	Measurement service port
proto	UDP		false	false	Measurement transport protocol
kbps	2000		false	false	Measurement bitrate
mtu	1472		false	false	Measurement max packet size
rcv_log_file	/tmp/rcv_log_file		false	false	Log file path

4.2.2 Basic performance evaluation (BPE) campaign

The second campaign is defined to evaluate the performance of the access network on a regular basis over a long time period. As shown in Tab. 4.8, its experiments are very similar to the ones defined for the CPE campaign, with the addition of TCP throughput measurements. It is worth to notice that the scripts associated to those experiments are the same as the ones defined in the previous section. To optimize the amount of information exchanged between the client and the server, such scripts are cached on the client side, so that only the input parameters are requested every time to the management server. This allows to keep the possibility to vary the parameters on each execution (e.g. the imposed generation throughput may be adjusted according to previous results).

Table 4.8: Experiments part of the periodic basic performance evaluation campaign.

ID	Measured metrics	Required bandwidth (kbps)	Duration (sec)	Measurement class
#1	Latency, Jitter & Packet loss	512	30	LIGHT
#2	Upstream UDP throughput	2000	15	INVASIVE
#3	Downstream UDP throughput	20000	15	INVASIVE
#4	Upstream TCP throughput	2000	15	INVASIVE
#5	Downstream TCP throughput	20000	15	INVASIVE

With respect to the CPE campaign, it provides a more complex schedule for its experiments. As reported in compact form in Tab. 4.9, we request the execution of each experiment a certain number of times for five different daytime periods, which represent peak hours and offload hours, and separately for each day of the week. This allows us to have enough measurements representative for each of these time periods. This campaign

terminates when all the requested repetitions are executed for each time period.

Table 4.9: Compact view of the schedule defined for the experiments.

DoW	DoM	MoY	TimeStart	TimeStop	Repetition	FilterFuncion
lun-dom			00:00:00	07:00:00	84	
lun-dom			07:00:00	12:00:00	60	
lun-dom			12:00:00	15:00:00	36	
lun-dom			15:00:00	20:00:00	60	
lun-dom			20:00:00	24:00:00	84	

4.2.3 BitTorrent performance evaluation (BTPE) campaign

The third and last campaign currently defined is dedicated to the measurement of BitTorrent [111] performance over a selected set of clients. The goal of such campaign is to establish if ISPs treat differently the traffic associated with such application. As shown in Tab. 4.10, the campaign is assigned only to connections having a downstream capacity larger than 2 Mbps. The experiments provided by this campaign, as shown in Tab. 4.11, are again based on the same scripts as in the other ones, since we emulate the generation of BitTorrent traffic generating it towards the default transport protocol port associated with such protocol (i.e. 6881).

Table 4.10: BitTorrent performance evaluation campaign selection criterion.

Name	Value	Param	IsFunction	Post	Description
Bandwidth > 2 M	cfPlan	{;;>2}	true	false	Selects only connections having download bandwidth more than 2 Mbps

Table 4.11: Experiments part of the BitTorrent performance evaluation campaign.

ID	Measured metrics	Required bandwidth (kbps)	Duration (sec)	Measurement class
#1	Latency, Jitter & Packet loss	512	30	LIGHT
#2	Upstream UDP throughput	2000	15	INVASIVE
#3	Downstream UDP throughput	20000	15	INVASIVE
#4	Upstream TCP throughput	2000	15	INVASIVE
#5	Downstream TCP throughput	20000	15	INVASIVE

4.3 Challenges and solutions

In this section we describe the most relevant challenges we had to face for the HoBBIT platform to work properly when deployed on a large scale.

4.3.1 Platform scalability

One of the main objectives of the HoBBIT platform is to reach a big number of client installations, in order to obtain statistically significant statistics about broadband access networks at fine grained aggregation levels. This a is really important aspect because, as explained in Sec. 3.1, the host-based approaches are affected by many unmanageable factors.

Scaling to a big number of users determines some scalability issues, which mainly affect the management server load and the scheduling algorithm (see Sec. 4.1) adopted to assign measurement requests to specific measurement servers.

In the following sections we present a scalability analysis of most critic resources managed by HoBBIT .

Scalability analysis of the scheduling algorithm

As explained in Sec. 4.1, the scheduling algorithm adopted by the management server for assigning resources to specific clients partitions the capacity of measurement servers in slices. Each experiment requires a certain amount of slices to be executed, thus the management server is responsible for choosing which server to select and when the client is allowed to perform the experiment. Indeed, if the required amount of slices is not available on any server, it is reserved for a future execution on the server which will have it first.

In order to validate the effectiveness of such algorithm in improving the scalability of the platform, we conducted some experiments by emulating experiment requests towards the real management server. One of the main goals of such analysis is to establish how many measurement servers are needed to support a certain amount of clients given the campaigns defined in Sec. 4.2.

In the following paragraphs we describe the methodology adopted and the results obtained by conducting such experiments.

Methodology. To evaluate the scalability of the above mentioned algorithm, we created an emulator of experiment requests generated by HoBBIT client. Written as a simple PHP script, such emulator is programmed to send requests for different experiment lists to the management server over a specified time period, thus emulating the presence of more real clients. This simple approach forces the execution of the algorithm, which allocates the available slices without the need to execute the experiments. To keep track of all the operations, the script generates a log file containing, for each request, the information necessary for our analysis: the request timestamp, the `FREE_TS` value after the allocation of the experiment, and the *waiting time* assigned to the client before starting the experiment.

To be conservative, in all the experiments we consider each measurement server to have 180 slices of 512 kbps, for a total usable capacity of about 92 Mbps over the theoretical 100 Mbps.

Simultaneous LIGHT experiments. In this first experiment we configured the emulator to generate 20 simultaneous requests per second for a LIGHT measurement lasting 30 seconds and requiring just one slice, for a total of 2000 requests. As provided by the algorithm, for each experiment the duration is considered augmented by 3 seconds to avoid border effects, which may cause interference between consecutive experiments.

As reported in Fig. 4.21, we executed the emulator by considering first one measurement server (red line) and then two servers (blue line). The two graphs respectively report the `FREE_TS` value and the *waiting time* as affected by the reception of subsequent requests.

When relying only on one measurement server (i.e. having only 180 available slices) the waiting time increases linearly, with an increment of 33 seconds every 180 requests, reaching a maximum of 250 seconds. When working with two servers (i.e. doubling the slices availability) the waiting time increases again linearly, but with increments of about 10 seconds, which is about one third of the previous case. This happens because, having more available slices, the slices already assigned are less frequently reassigned, thus keeping their `FREE_TS` value smaller. The same behavior is analyzed about the `FREE_TS` value.

From this simple experiment we were able to establish that doubling the number of the servers increases the scalability by a factor greater than 2.

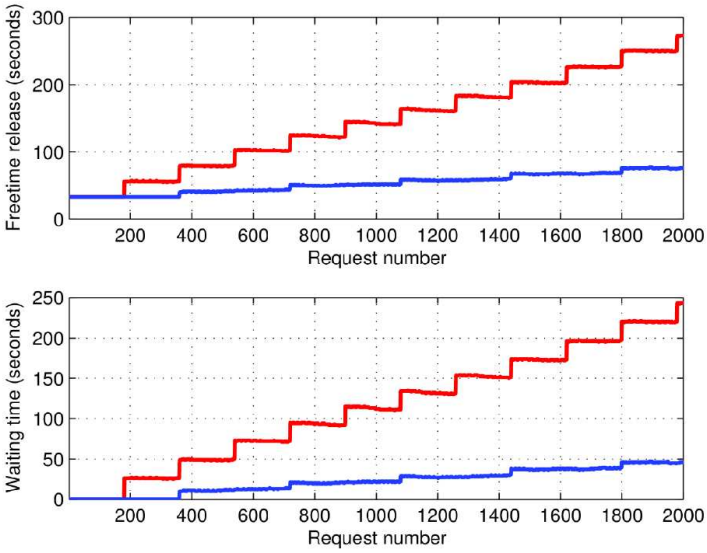


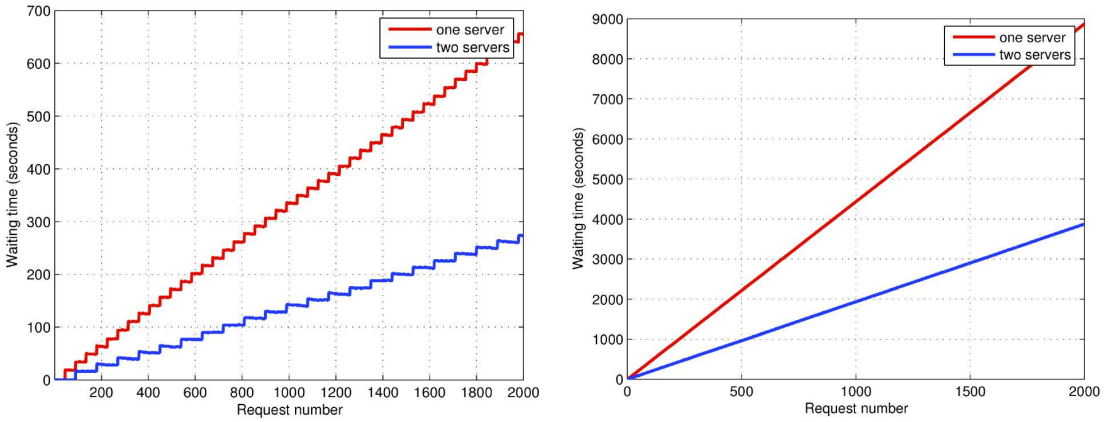
Figure 4.21: Scalability with simultaneous LIGHT requests. Time to wait for executing an experiment when receiving 20 simultaneous requests per second.

Simultaneous INVASIVE experiments. In this second experiment we configured the emulator to generate again 20 simultaneous requests per second for an INVASIVE measurement lasting 15 seconds. We executed the measurement into two different configurations, in order to be respectively equivalent to an upload and a download throughput experiment. In the first case, the measurement requires just 4 slices, while in the second 40. Also here we repeated both measurements with one and two measurement server.

Fig. 4.22(a) shows the results obtained when requesting the upload throughput measurement. Working with only one measurement server the waiting time is incremented by 18 seconds on every 45 requests, obtaining a maximum waiting time of 650 seconds (i.e. about 11 minutes). When having an additional server the waiting time is incremented only every 90 requests, reaching a maximum waiting time of 280 seconds (i.e. less then 5 minutes).

Fig. 4.22(a) shows a similar trend for the download throughput measurement, but one order of magnitude bigger. Indeed, with one server the maximum waiting time reached is of 150 minutes with one server and of 66 minutes with two servers.

As expected, with respect to the LIGHT experiment these cases are more critical, since they occupy many slices on each request. Hence, adding more servers helps in reducing the waiting times, but with a smaller effect.



(a) Upload throughput measurement.

(b) Download throughput measurement.

Figure 4.22: Scalability with simultaneous INVASIVE requests. Time to wait for executing an experiment when receiving 20 simultaneous requests per second.

Simultaneous CPE campaigns. After the evaluation of the scalability of the scheduling algorithm in presence of only one type of request, we made a step towards a more realistic case. We configured the emulator for generating requests for lists of experiments as defined by the CPE campaign, which means that each emulated client serially requests three different experiments corresponding to the following sequence:

1. roud-trip time, jitter, and packet loss measurement lasting 30 seconds and requiring 1 slice;
2. upload throughput measurement lasting 15 seconds and requiring 4 slices;
3. download throughput measurement lasting 15 seconds and requiring 40 slices.

We configured the emulator to generate such kind of requests 20 times per seconds, for a total of 2000 campaign requests, which correspond to a total of 6000 experiments.

To help in comparing the results with the previous analysis Fig. 4.23 only shows the waiting time as a function of the first 2000 requests.

Being the set of experiments heterogeneous the waiting time still increases linearly, but at a smaller rate then with respect to the previous analysis. Particularly, the advantage obtained when using two servers is still more than double.

Poissonian connection parameters estimation experiments. All the previous analysis were considered generating requests following an highly unrealistic pattern. In-

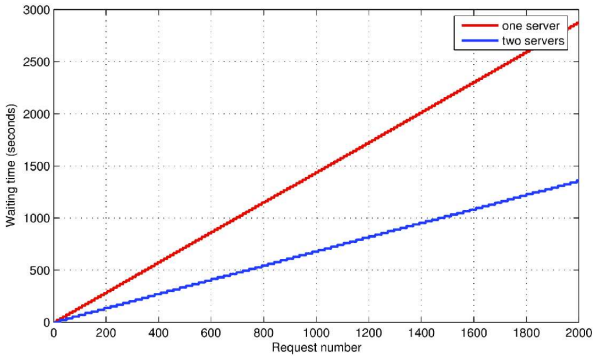


Figure 4.23: Scalability with simultaneous CPE requests. Time to wait for executing an experiment when receiving 20 simultaneous requests per second.

deed, in it is very improbable that 20 clients spontaneously generate requests at the same time and that requests are received at a constant rate. We modified then the emulator script to generate CPE requests following a poisson distribution.

Hence, we try to emulate the condition in which 500 HoBBIT clients are installed for the first time in a period of one hour. The script acts for each emulated client according to the following steps:

1. waits for an exponentially distributed random number of seconds;
2. sends the request for the experiment #1 to the management server;
3. waits for the duration of the experiment #1 plus 3 seconds;
4. sends the request for the experiment #2 to the management server;
5. waits for the duration of the experiment #2 plus 3 seconds;
6. sends the request for the experiment #3 to the management server;
7. waits for the duration of the experiment #3 plus 3 seconds.

Fig. 4.24 shows the results obtained with one and two servers. We observed that a single server is not able to properly manage the load provided, since the waiting time grows indefinitely over time. On the other hand, having two servers allows to keep the waiting time under 100 seconds all the time.

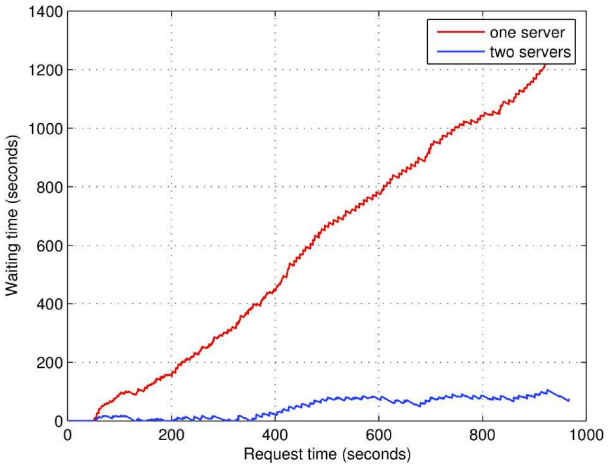
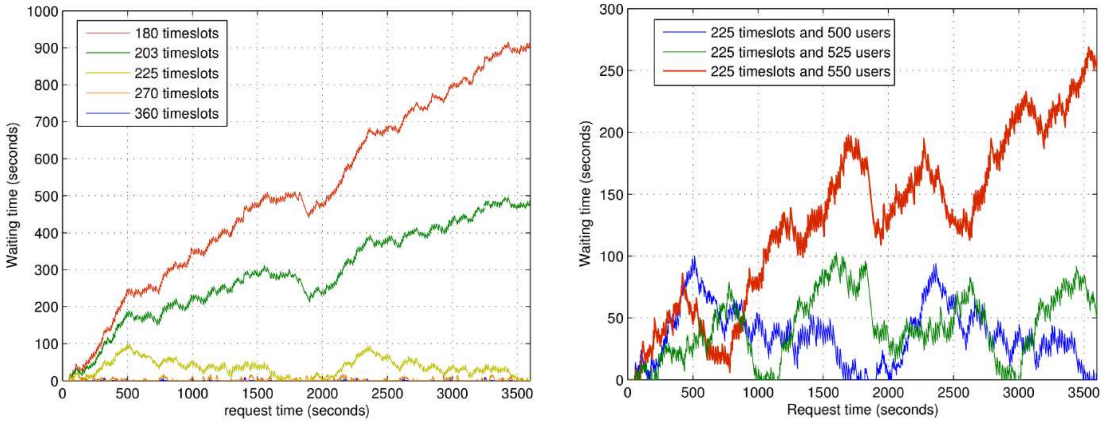


Figure 4.24: Scalability with poissonian CPE requests. Time to wait for executing an experiment when receiving 500 poissonian requests.

Periodic poissonian connection parameters estimation experiments. Despite the previous experiment considers a more realistic case it does not consider the periodicity which characterizes the normal behavior of a HoBBIT client. In this last experiment we consider the case in which each emulated client requests a CPE campaign every 30 minutes, which is the default interval used by HoBBIT to request a list of experiments. Also in this case we consider 500 clients generating requests according to a poisson distribution.



(a) Single requests from 500 users having increasing available slices.

(b) Periodic requests from up to 550 users having 225 available slices.

Figure 4.25: Time to wait for executing an experiment when receiving poissonian CPE requests.

The experiment has been conducted by operating a binary search for the instability

Table 4.12: Management server average response time to experiment requests.

Experiment	Total request time (μs)	DB Query time (μs)	PHP processing time (μs)
Latency, Jitter & Packet loss	0.036	0.024	0.012
Upstream throughput	0.038	0.026	0.012
Downstream throughput	0.039	0.027	0.012

threshold among the one server (180 slices) and two servers (360 slices) case. As shown in Fig. 4.25(a) the growth of the waiting time becomes unstable when having less than 225 slices. Hence, fixing the number of slices to 225, we tried then to increment the number of clients by the 5% and the 10%, in order to find another instability threshold. As reported in Fig. 4.25(b) with 225 slices the system remains stable when having 525 users and becomes unstable with 550 users.

Management server response time analysis

Another factor to consider about the scalability of the system is bound to the frequent number of queries received by the management server. Hence, by using the emulator script used in the previous analysis, we evaluated the load generated on the Web server when working in the worst case considered by the latest scalability experiment (i.e. 550 clients and 225 slices).

Hence, we considered the difference in time between each reply received and request sent, by obtaining the average delay values reported in Tab. 4.12. We observed that most of the delay is due to the database querying process and it is mostly independent on the experiment type.

For sake of completeness, we report in Fig. 4.26 the timeseries of the delays obtained over 160 K requests during such experiment.

4.3.2 Real-time reporting at different aggregation levels

Once experimental data is stored into the database, it is important to be able to infer relevant statistics from it. Having a complex database structure, as reported in 4.1.1, makes it difficult to obtain in realtime when the amount of data is consistent. In details, one of the main goals of the HoBBIT project is to make public accessible performance statistics at different aggregation levels, by considering geographical locations (i.e. postcodes, municipalities, provinces, regions), access network properties (e.g. ISP, Plan, IP block,

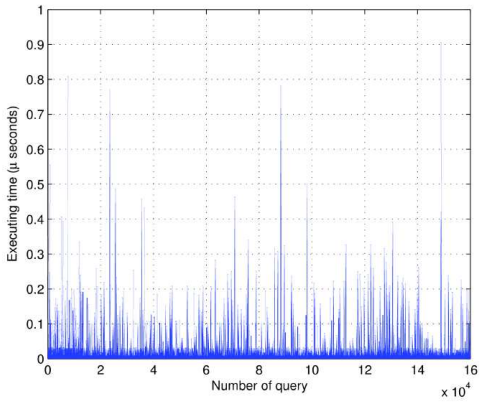


Figure 4.26: Management server response time over 160 K requests.

access technology, ...) and variable time intervals (e.g. hour, day, week, month, ...).

In order to obtain such aggregations, the queries performed on the database need to execute several JOIN operations, which became really slow when operating on big tables. Thus, some optimizations are required both in the database structure and the database management system (DBMS) configuration.

In the following section we describe how we implemented the support for materialized views in PostgreSQL and how we applied them to the HoBBIT database, validating their effectiveness with some experiments.

Implementing optimizations in PostgreSQL

Since HoBBIT platform utilizes a PostgreSQL database, after evaluating the possible optimizations to adopt for improving querying performance, we evaluated how to implement them in that context.

Arrays. PostgreSQL allows columns of a table to be defined as variable-length multidimensional arrays of built-in or user-defined base, enum, or composite type. To illustrate the use of array types, we create this table:

```
CREATE TABLE salary (
   name     text,
   pay      integer[],
   schedule text[][]
);
```

As shown, an array data type is named by appending square brackets ([]) to the data type name of the array elements. The previous command will create a table named

salary with a column of type text(name), a one-dimensional array of type integer (pay), which represents the employee's salary by quarter, and a two-dimensional array o of text (schedule), which represents the employee's weekly schedule.

An alternative syntax, which conforms to the SQL standard by using the keyword ARRAY, can be used for one-dimensional arrays:

```
pay integer ARRAY,
```

Single elements of an array can be accessed as in the following query example, which retrieves the names of the employees whose pay changed in the second quarter:

```
SELECT name
FROM salary
WHERE pay[1] <> pay[2];
```

The array subscript numbers are written within square brackets. By default PostgreSQL uses a one-based numbering convention for arrays, that is, an array of n elements starts with array[1] and ends with array[n].

Materialized Views. PostgreSQL, unlike other commercial DBMS, does not provide built-in support for materialized views. However, this support can be added by creating a physical table, populated with the result of a query, and properly setting stored procedures and triggers necessary to keep it up to date. This allows to implement a variety of materialization techniques that are not available in other DBMSs.

There are four techniques to be implemented for materialized views:

- *Snapshot*: creates a physical table as the result of selecting everything out of a view, and refreshing it at a given interval;
  - pro: easy to set up;
  - con: gets out of sync quickly.
- *Very Lazy*: like snapshot, but only out of sync rows get updated at refresh time: requires keeping track of which rows are out of sync;
  - pro: lighter refresh than snapshot;
  - cons: still gets out of sync quickly and needs an auxiliary table to implement.

- *Lazy*: starts with a snapshot, refreshes rows that are out of sync at the end of each transaction;
  - pros: always in sync, only affected rows are updated;
  - con: there is no after-transaction trigger in PostgreSQL.
- *Eager*: like Lazy, but updates materialized view after each statement. Uses triggers after update, insert, and delete on all referenced tables;
  - pro: always in sync;
  - con: bad in one-to-many relationships updates.

Since we need data to be consistent we focus our attention on the eager technique, because currently there is no way to put a trigger when transaction is committed to implement the lazy technique.

Eager materialized views. An eager materialized view is updated whenever the view changes and this is done with a system of triggers on all the underlying tables. It is important to consider what actually makes up the data in a materialized view, or where the data in the view comes from. If the view definition relies on other views, then we have to consider all the tables that compose that view as well. It is also crucial to consider how the data in the underlying table relates to or affects the data in the materialized view.

Taking into account the previous aspects, to implement a materialized view using the eager technique it is necessary to define two functions and several triggers. The first function allows to refresh a single row of the materialized view and can be created following these guidelines:

- identify the primary key of the materialized view and, if not present, redefine it to create one;
- define the function to take as argument that primary key;
- let the function delete the row with that primary key;
- let it select the row with the same key from the original view and insert it into the materialized view;

The second function operates calling the first one only on the rows that have been affected by the last change. Finally, triggers should be created for every action on every underlying table (i.e. INSERT, UPDATE, and DELETE), following these guidelines:

- identify the primary key value(s) for the materialized view of the affected rows;
- if there is a many-to-one or many-to-many, many rows in the materialized view will be affected;
- for DELETE and INSERT, merely call the first refresh function for each primary key value;
- for UPDATE, identify whether the update is going to change which row(s) in the materialized view which this row will affect;
- if so, then you will have to refresh rows for both the old and new values;
- otherwise, only the old or new values will do;
- apply the triggers so that they are called after the operation is performed.

Table partitioning. PostgreSQL supports basic table partitioning, and the procedure to set up a partitioned table can be done according to the steps described in the following:

- create the "master" table, from which all the partitions will inherit, which will contain no data;
- create several "child" tables (henceforth partitions) that each inherit from the master table, which normally do not add any columns to the set inherited from the master;
- add table constraints to the partition tables to define the allowed key values in each partition[2], as reported in the following example;

```
CREATE TABLE measurement_2011_11 (
    CHECK ( date >= DATE '2011-11-01'
        AND date < DATE '2011-12-01' )
) INHERITS (measurement);
```

[2]Constraints have to guarantee no overlap between key values permitted in different partitions.

- For each partition, create an index on the key column(s), as well as any other indexes you might want;
- Define a trigger to redirect data inserted into the master table to the appropriate partition, as shown in the following example;

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO measurement_y2008m01 VALUES (NEW.*);
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER insert_measurement_trigger
    BEFORE INSERT ON measurement
    FOR EACH ROW EXECUTE PROCEDURE measurement_insert_trigger();
```

- Ensure that the `constraint_exclusion` configuration parameter is not disabled in `postgresql.conf`, to allow queries to be properly optimized.

By setting the trigger as explained before, we have to redefine the trigger function each month so that it always points to the current partition. Another solution could be to insert data and have the server automatically locate the partition into which the row should be added, by defining a more complex trigger function (see Fig. 4.27).

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF ( NEW.logdate >= DATE '2006-02-01' AND
         NEW.logdate < DATE '2006-03-01' ) THEN
        INSERT INTO measurement_y2006m02 VALUES (NEW.*);
    ELSIF ( NEW.logdate >= DATE '2006-03-01' AND
            NEW.logdate < DATE '2006-04-01' ) THEN
        INSERT INTO measurement_y2006m03 VALUES (NEW.*);
    ...
    ELSIF ( NEW.logdate >= DATE '2008-01-01' AND
            NEW.logdate < DATE '2008-02-01' ) THEN
        INSERT INTO measurement_y2008m01 VALUES (NEW.*);
    ELSE
        RAISE EXCEPTION 'Date out of range';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

Figure 4.27: Complex trigger function for table partitioning.

Optimizing the HoBBIT 's database

As described in Sec.4.1.1, the HoBBIT 's database is designed to store measurement results conducted by every client installed. In detail, `Samples` and `MeasureOutputs` tables tend to grow quickly over time and can easily reach millions of rows. Since all the statistics we want to extract are based on such tables, we applied the optimization techniques described in the previous sections to them.

Storing measurement results with array columns. According to the initial database design, each experiment returning a set of samples for a measured parameter populates both the `MeasureOutputs` and `Samples` tables. Thus, to calculate statistics on a set of measurement results, we need to execute a JOIN on the tables. This condition is detrimental for performance when executing queries and should be avoided.

```
CREATE TABLE "MeasureOutputs"
(
  "Timestamp" timestamp with time zone,
  "IDOutput" integer NOT NULL DEFAULT
             nextval('"Measureoutputs_IDOutput_seq"'::regclass),
  "Parameters" text,
  "IDConnection" integer,
  "ClientIp" inet,
  "IPServer" inet,
  "UUID" uuid,
  "OutputNumber" smallint,
  "IDExperiment" integer,
  "ParametersValue" character varying,
  "Samples" double precision[],
  "Stats" double precision[],
  CONSTRAINT pk PRIMARY KEY ("IDOutput")
);
```

Figure 4.28: MeasureOutputs table with array columns.

To avoid the JOIN operation we removed the `Samples` table and modified the `MeasureOutputs` table, by adding to it an array column for the samples. After this simple optimization, to obtain statistics aggregated at higher layers (e.g. per connection), we still need to compute them starting from samples. For example, if 100 experiments have been conducted for a specific connection and we want to compute the average value of its results, where each measurement returns 20 samples, we need to execute the calculation over 2000 samples. Thus, to further improve the performance we added another array column to store basic statistics pre-calculated on samples, so that in the previous example only 100 values have to be considered. The resulting `MeasureOutputs` schema is reported in Fig.4.28

and every time a new experiment is executed both the samples returned and the statistics (e.g. minimum, maximum, average, ...) are inserted into the respective array columns.

Materializing views for quick access to aggregated statistics. As shown in Sec.4.1.1, the core of the HoBBIT database is represented by the `Connections` table, which is associated to most other tables. For instance, the only way to obtain statistics on a parameter measured by a specific experiment, aggregated by geographical region, is to execute a JOIN among the `MeasureOutputs`, `Connections` and `Regions` tables. The same for any other kind of aggregation. Thus, many JOIN operations may be expected involving the `MeasureOutputs` and the `Connections` tables.

To provide statistics aggregated at different levels we designed a set of views as reported in Fig.4.29 for the geographical case.

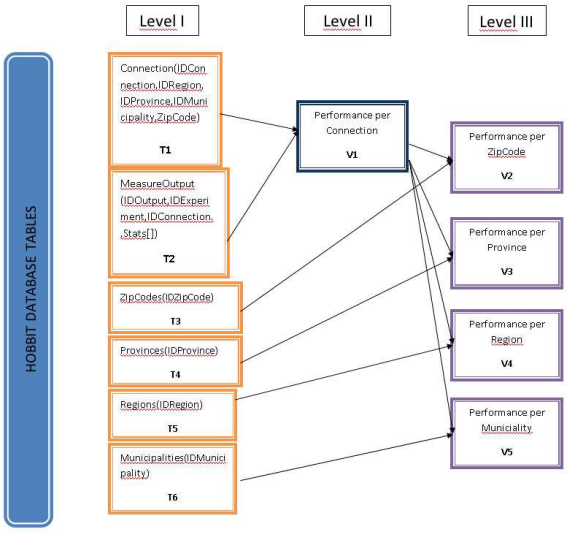


Figure 4.29: Views proposed for geographical aggregation of statistics.

As shown in figure, all the views on the third level depend on the `PerformancePerConnection` view, which is defined as reported in Fig. 4.30. Hence, we decided to make it a materialized view using the eager technique. To create the initial materialized view, we execute the following SQL command:

```
CREATE TABLE "PerformancePerConnectionM" AS
SELECT *
FROM "PerformancePerConnection";
```

It populates a new physical table with a snapshot of the previous view. At this point on the database there are two views: the original and the materialized one. Then, we need to implement the refresh function which will keep the materialized table up to date when new entries are added to both `Connections` and `MeasureOutputs` tables. The refresh function requires two parameters: `IDCon` and `IDExp`, which are used to select the new rows added to `MeasureOutput` table and to insert them into the materialized view, and is defined as reported in Fig. 4.31. It is worth to notice that if we want to refresh the rows, we must fist delete previously existing entries, in order to avoid duplicates.

When a new record is inserted in `MeasureOutputs`, we want that record to be added to the materialized view along with an automatic update of statistics such as min, max and average. The effect is that every new row added to `MeasureOutputs` makes an automatic insert into `PerformancePerConnectionM`, which is equivalent to an update of connection statistics. To automatically refresh the materialized view as changes are made to underlying tables, we need to create some triggers on them. Since `PerformancePerConnectionM` does not have a primary key, we create a key composed of `IDConnection` and `IDExperiment` on it. According to the HoBBIT specifications, the underlying tables will only be mod-

```
CREATE OR REPLACE VIEW "PerformancePerConnection" AS
 SELECT "Connections"."IDConnection", "MeasureOutputs"."IDExperiment",
     min("MeasureOutputs"."Stats"[1]) AS min,
     max("MeasureOutputs"."Stats"[2]) AS max,
     avg("MeasureOutputs"."Stats"[3]) AS avg
 FROM "Connections"
   LEFT JOIN "MeasureOutputs" ON "MeasureOutputs"."IDConnection" = "Connections"."IDConnection"
   GROUP BY "Connections"."IDConnection", "MeasureOutputs"."IDExperiment"
   ORDER BY "Connections"."IDConnection";
```

Figure 4.30: Definition of the PerformancePerConnection view.

```
CREATE OR REPLACE FUNCTION refresh_PerformancePerConnectionM(IDCon integer, IDExp integer)
returns void
security definer
language 'plpgsql' as $$
begin
 INSERT INTO "PerformancePerConnectionM"
 SELECT *
 FROM "PerformancePerConnection"
 WHERE "PerformancePerConnection"."IDConnection" = IDCon AND
       "PerformancePerConnection"."IDExperiment" = IDExp ;
 DELETE
 FROM "PerformancePerConnectionM"
 WHERE "PerformancePerConnection"."IDConnection" = IDCon AND
       "PerformancePerConnection"."IDExperiment" = IDExp ;
end
$$;
```

Figure 4.31: PerformancePerConnectionM refresh function definition.

ified by INSERT INTO operations, thus we only need to setup INSERT INTO triggers. To properly implement the triggered action we defined a support function as reported in Fig. 4.32.

```
CREATE OR REPLACE FUNCTION support_insert_trigger() returns trigger
security definer
language 'plpgsql' as $$
BEGIN
PERFORM refresh_PerformancePerConnectionM(new.IDConnection,new.IDExperiment);
return null;
END
$$;
```

Figure 4.32: PerformancePerConnectionM trigger support function definition.

It is worth to notice that by using the PERFORM keyword we explicitly avoid to store the result of a select statement. For each function, we also need to add a corresponding trigger to the table itself, so that the database knows which method to call when a particular event occurs. The SQL command in Fig. 4.33 creates a trigger after and INSERT INTO operation on the `MeasureOutputs` table.

```
CREATE TRIGGER trg_MeasureOutputs AFTER INSERT ON MeasureOutput
FOR EACH ROW EXECUTE PROCEDURE support_insert_trigger();
```

Figure 4.33: MeasureOutputs trigger definition.

After applying the eager materialization technique to the `PerformancePerConnection` view, we investigated on how to further optimize the materialized view. We found that the refresh function can be properly modified in several points in order to avoid costly processing. Indeed, the current refresh function includes a SELECT on the original view, which still requires to join the underlying tables on each execution. This means that we are constantly re-running a very expensive query.

To avoid this we have to create a mechanism that calculates maximum, minimum and average according to the data already present in the materialized view. In other words, we can update the maximum, minimum and average statistics without relying on the original view.

Recalling that the refresh function is invoked when a new entry is inserted into the `MeasureOutputs` table, we identified two different cases which require different operations with respect to a specific `(IDExperiment, IDConnection)` couple:

1. there is no previous entry in the table;

2. there are previous entries in the table.

In the first case, the refresh function just needs to select from the new tuple some fields and to insert them into the materialized view (i.e. `IDConnection`, `IDExperiment`, `IDOutput`, and `Stats[]`). To do this we had to modify the trigger support function in order to pass also the `IDOutput` field to the refresh function, as shown in Fig. 4.34. Hence, to manage this first case, the refresh function has been modified to perform only a INSERT INTO operation on the materialized view, as shown in Fig. 4.35, where the constant value 1 represents the initial number of `MeasureOutputs` entries having that `(IDExperiment, IDConnection)` couple.

```
CREATE OR REPLACE FUNCTION support_insert_trigger() returns trigger
security definer
language 'plpgsql' as $$
BEGIN
PERFORM refresh_PerformancePerConnectionM(new.IDOutput,new.IDConnection,new.IDExperiment);
return null;
END
$$;
```

Figure 4.34: Modified MeasureOutputs trigger support function definition.

```
INSERT INTO "PerformancePerConnectionM"
SELECT "MeasureOutputs"."IDConnection","MeasureOutputs"."IDExperiment",
       "MeasureOutputs"."Stats"[1],
       "MeasureOutputs"."Stats"[2],
       "MeasureOutputs"."Stats"[3], '1'
FROM "MeasureOutputs"
WHERE "MeasureOutputs"."IDOutput" = IDOut ;
```

Figure 4.35: Query to update the materialized view in the first case.

In the second case, being already present other `MeasureOutputs` entries having the same `(IDExperiment, IDConnection)` couple as the new entry, also the materialized view already contains a row corresponding to it. Thus, the refresh function in this case has to update such row, which means that only the columns related to the statistics have to be refreshed. Hence we extract the statistics from the new entry using an SELECT INTO query which stores them respectively in the `NewMin`, `NewMax`, and `NewAvg` variables (see Fig. 4.36). Once these variables have been extracted, the refresh function has to process the respective fields from the materialized view in order to properly update them. By using a SELECT INTO we store them into the `OlnMin`, `OldMax`, and `OldAvg` variables. While current minimum and maximum statistics can be obtained by simply comparing old

and new variables, the current average is calculated according to the following equation:

$$CurrAvg = \frac{(OldAvg * OldAvgCnt) + NewAvg}{OldAvgCnt + 1} \tag{4.5}$$

which computes a weighted average of new and old values. The `OldAvgCnt` variable correspond to the current number of `MeasureOutputs` entries having the same `(IDExperiment, IDConnection)` couple as the new entry. To avoid further processing, this variable is stored as a column into materialized view.

Finally, the refresh function has to be able to distinguish the first and the second case. To implement this, its first statement tries to extract the minimum (`OldMin`) from the materialized view. It behaves as in the first case if it is null, and as in the second case otherwise. The complete optimized version of the refresh function is shown in Fig. 4.37.

Partitioning MeasureOutputs table for maintaining good long-term performance. Since the `MeasureOutputs` table tends to grow quickly over time, we decided to apply a time-based partition on it. In order to be able to adjust the partitioning over time, we decided to apply a hierarchical partitioning approach in which we initially partition the table by month. Whenever the HoBBIT user base becomes so large that a single partition results to be still too big, we partition it per week. Beyond that, if necessary, we can start partitioning each week by geographical locations, starting from the region level and so on.

We are currently working on implementing such feature as a procedure that automatically reacts to the conditions of the database.

Validation experiment

By considering the optimizations proposed to reduce the delay of queries on aggregated statistics, we performed a performance comparison in the following three cases:

A. Original database structure with separated `MeasureOutputs` and `Samples` tables;

```
SELECT INTO NewMin, NewMax, NewAvg
       "MeasureOutputs"."Stats"[1],
       "MeasureOutputs"."Stats"[2],
       "MeasureOutputs"."Stats"[3]
FROM  "MeasureOutputs"
WHERE "MeasureOutputs"."IDOutput" = IDOut ;
```

Figure 4.36: Query to extract statistics from MeasureOutputs.

```
CREATE OR REPLACE FUNCTION refresh_PerformancePerConnectionM(IDOut integer, IDCon integer, IDExp integer)
RETURNS void AS
$BODY$
declare
  OldMin    double precision;
  OldMax    double precision;
  OldAvg    double precision;
  OldAvgNum double precision;
  NewMin    double precision;
  NewMax    double precision;
  NewAvg    double precision;
  CurrMin   double precision;
  CurrMax   double precision;
  CurrAvg   double precision;
begin
  SELECT INTO OldMin "PerformancePerConnectionM"."min"
  FROM "PerformancePerConnectionM"
  WHERE "PerformancePerConnectionM"."IDConnection" = IDCon AND
"PerformancePerConnectionM"."IDExperiment" = IDExp ;

  IF OldMin is null THEN
     INSERT INTO "PerformancePerConnectionM"
     SELECT "MeasureOutputs"."IDConnection", "MeasureOutputs"."IDExperiment",
    "MeasureOutputs"."Stats"[ 1], "MeasureOutputs"."Stats"[2],"MeasureOutputs"."Stats"[3],'1'
     FROM "MeasureOutputs"
     WHERE "MeasureOutputs"."IDOutput" = IDOut ;
  ELSE
     SELECT INTO OldMax, OldAvg, OldAvgNum
       "PerformancePerConnectionM"."max",
       "PerformancePerConnectionM"."avg",
       "PerformancePerConnectionM"."numavg"
     FROM "PerformancePerConnectionM"
     WHERE "PerformancePerConnectionM"."IDConnection" = IDCon AND
   "PerformancePerConnectionM"."IDExperiment" = IDExp ;

     SELECT INTO NewMin, NewMax, NewAvg
"MeasureOutputs"."Stats"[1],
"MeasureOutputs"."Stats"[2],
"MeasureOutputs"."Stats"[3]
     FROM  "MeasureOutputs"
     WHERE "MeasureOutputs"."IDOutput" = IDOut ;

     IF NewMin < OldMin THEN CurrMin = NewMin;
     ELSE CurrMin=OldMin;
     END IF;

     IF NewMax > OldMax THEN CurrMax = NewMax;
     ELSE CurrMax = OldMax;
     END IF;

     CurrAvg = ((OldAvg * OldAvgNum) + NewAvg)/(OldAvgNum + 1);

     UPDATE "PerformancePerConnectionM"
     SET "IDConnection" = IDCon, "IDExperiment" = IDExp, "min" = CurrMin, "max" = CurrMax,
 "avg" = OldAvg, "numavg" = OldAvgNum + 1
     WHERE "IDConnection" = IDCon AND "IDExperiment" = IDExp ;
  END IF;
end
$BODY$
LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 100;
```

Figure 4.37: Optimezed refresh function definition.

B. Database optimized by removing the `Samples` table and incorporating the samples into an array-type column;

C. Database further improved by materializing the PerformancePerConnection view.

In order to evaluate the query delay in such conditions, we populated the tables with artificial data. All the experiments were made evaluating the time needed to execute the following query:

```
SELECT * FROM "PerformancePerConnection";
```

We first considered the case in which an increasing number of HoBBIT clients is active for one month and performs the CPE campaign each 30 minutes. If considering a maximum of 250 clients, each of them associated to a different connection, this results in a total number of about 1 million `MeasureOutputs` rows. This case is representative of the growth of the user base we expect.

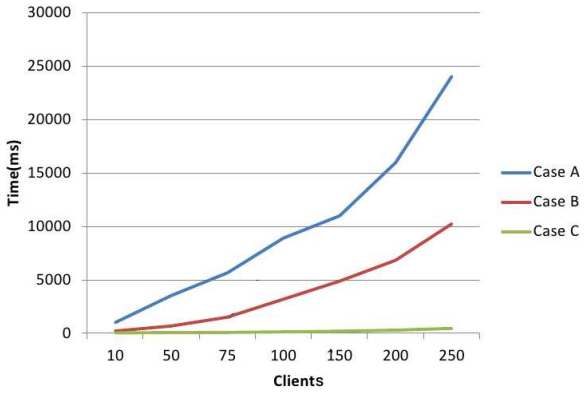


Figure 4.38: Query delay obtained with an increasing number of HoBBIT clients operating for one month.

Fig. 4.38 reports the comparison of the delays obtained in the cases A, B and C. As shown both the optimizations have a big effect on the performance, but only materializing the view allows to maintain acceptable delays.

4.4 A preliminary study of broadband in Italy from the hosts

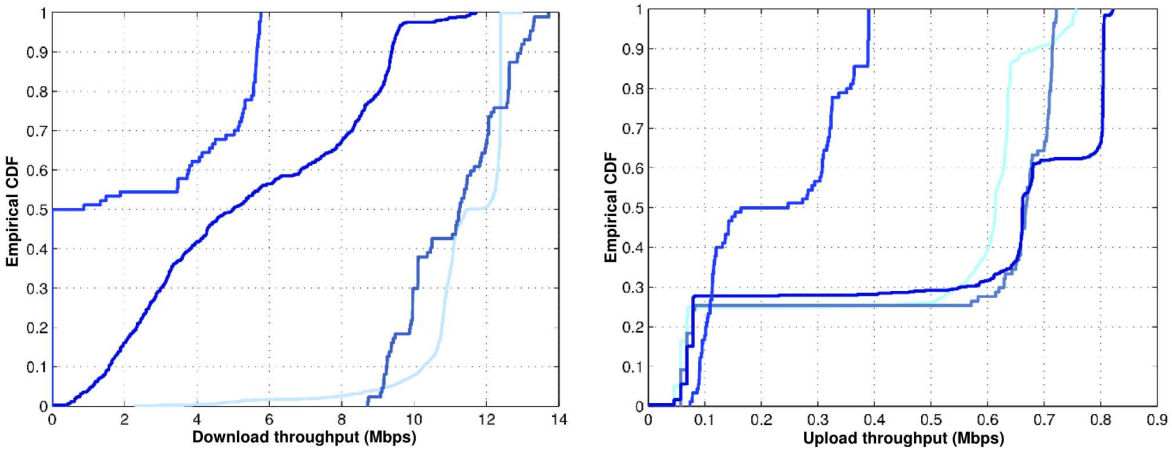
The current HoBBIT deployment counts about 100 clients in Italy, which totally performed thousands of measurements on as many connections and cover different service plans of major ISPs in Italy.

By selecting 4 ADSL connections having equivalent service plans (i.e. 20 Mbps downstream and 1 Mbps upstream capacity) from different ISPs and counting more than 100 experiments, we conducted a comparison of their perceived performance relying on data collected by both BPE and BTPE campaigns (see 4.2).

In the following sections we describe two preliminary analysis we conducted on the selected connections in order to compare their performance.

4.4.1 Basic performance evaluation

By exploiting the data collected by BPE campaigns, we conducted a preliminary analysis of the performance perceived by the four ADSL connections with respect to speed metrics (i.e. upload and download throughput) and metrics for interactive applications (i.e. latency and jitter).



(a) Download throughhput.

(b) Upload throughhput.

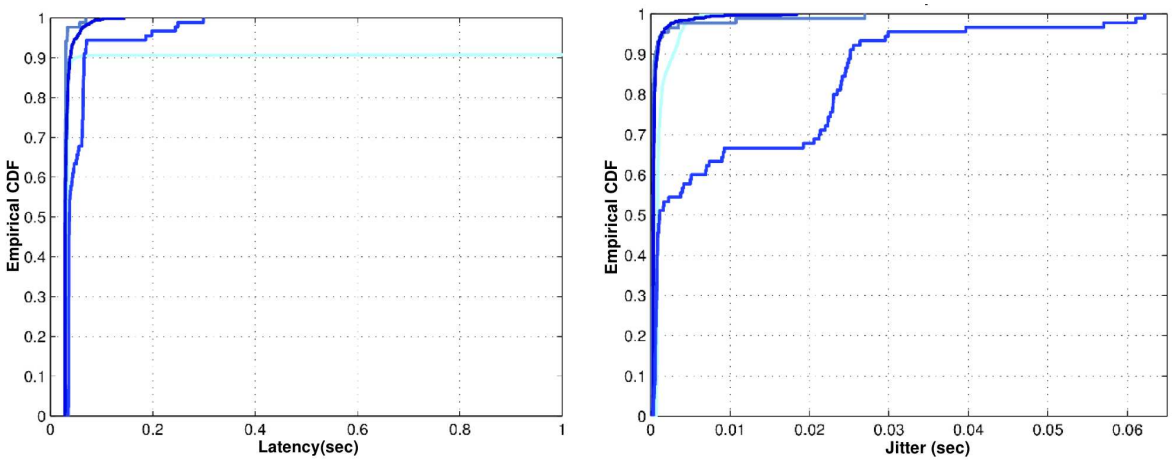
Figure 4.39: Speed metrics over 20 Mbps service plans.

We start by analyzing the speed metrics. Fig. 4.39(a) shows how download throughput is always below the advertised rate and never reaches values greater than 13 Mbps. All

the connections observe an high variability about this metric. We identify two extreme cases: one of them fails in executing the download in half cases (i.e. throughput equals to zero); another never experience values lower than 8.6 Mbps.

Fig. 4.39(b) shows how upload throughput is generically more consistent, but assuming a few different discrete values. We observe that three connections obtain values higher than 600 Kbps in more than the 70% of cases, while the remaining never experience values above 400 Kbps. Moreover, in the 30% of experiments all of them obtain values below 100 Kbps.

Passing to the analysis of metrics for interactive applications, looking at Fig. 4.40(a), we found latency to be very consistent for all the four ISPs. Only one connection experience in the 10% of cases values above one second. When considering jitter the situation is similar (see Fig. 4.40(b)), where another connection in more than 50% of cases experience high values.



(a) Latency. (b) Jitter.

Figure 4.40: Metrics for interactive applications over 20 Mbps service plans.

On one hand, as a result of this preliminary analysis, we conclude that similar service plans belonging to different ISPs may obtain significantly different speed performance, which in some cases are more stable than in others. Moreover, none of them achieve the advertised download throughput, which may be caused by the limitations of DSL technology (see Sec. 1.1.1). On the other hand, interactive applications experience similar performance with the exclusion of some specific cases. We underline that, being the measurements potentially affected by different kind of interference, the validity of such

conclusions should be verified on a bigger set of connections for each ISP, having equivalent service plans.

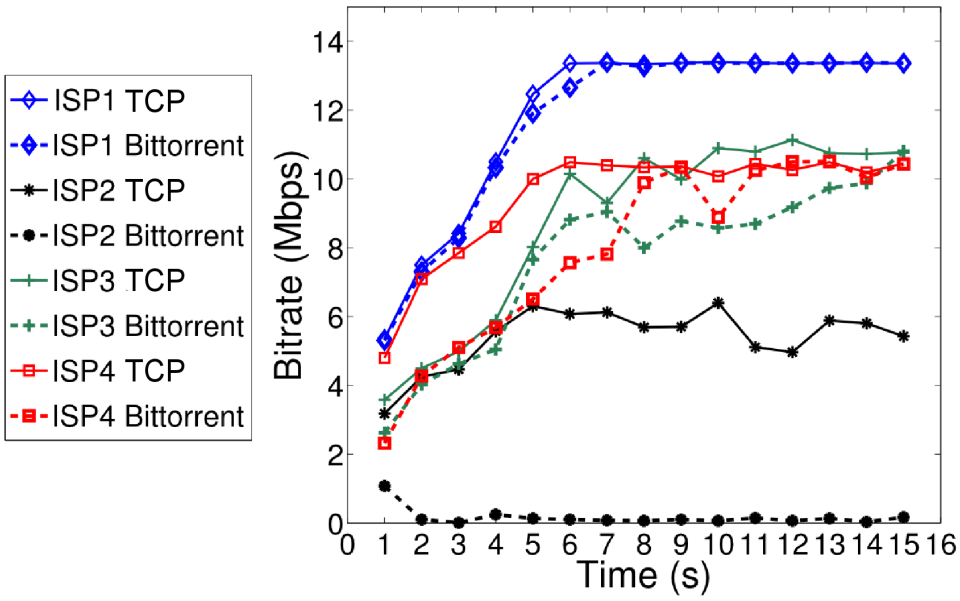


Figure 4.41: Downstream TCP throughput for four ISPs (20 Mbps service plans).

4.4.2 BitTorrent performance evaluation

By exploiting the data collected by both BTPE and BPE campaigns, we conducted a preliminary analysis of the performance perceived by the four ADSL connections with respect to the BitTorrent traffic.

Fig. 4.41 shows the results obtained by downstream TCP throughput measurements for all the ISPs (we refer to them with the terms ISP1, ISP2, ISP3 and ISP4) when evaluated by both the BPE and BTPE campaigns. Looking at this figure we can observe that one of the ISPs (i.e. ISP2) out of four is actually enforcing port based BitTorrent traffic shaping. Indeed, the throughput obtained when generating BitTorrent traffic is much lower than that observed when generating generic TCP traffic. Moreover, we notice that the initial TCP throughput achieved varies among different connections and, in one case (i.e. ISP4), between generic TCP and BitTorrent.

By analyzing the downstream UDP throughput for the same connections, as shown in Fig. 4.42, no discrimination are detected for BitTorrent traffic, but we still observe that none of the connections achieve the advertised rate, even if in this case performance is

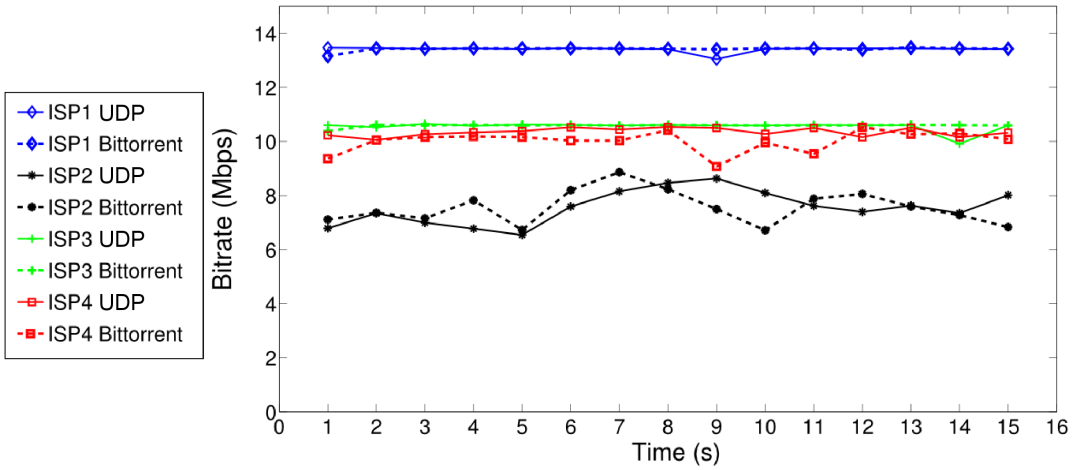


Figure 4.42: Downstream UDP throughput for four ISPs (20 Mbps service plans).

mostly stable over time.

Chapter 5

BISmark : adopting the router-based approach

Following the guidelines defined in Sec. 3.2, we developed a router-based architecture named BISmark [1] (Broadband Internet Service benchMARK), which has been realized in collaboration with the Georgia Institute of Technology of Atlanta. It requires the installation of a Linux-based wireless router in users' homes, which acts as the local network gateway and periodically conducts performance evaluation experiments towards measurement servers mainly hosted by the MLAB open platform [112].

The gateway sits in the home network directly behind the modem (e.g. cable, DSL, ...), performs measurements both to the *last mile router* (extracted as the first publicly routable IP hop on the path to the Internet) and to servers in the wide area, and sends their results back to a central repository for further analysis.

5.1 Architecture components

The architecture designed for BISmark includes all the basic components required by the ideal architecture defined in Sec. 3.2, where Measurement Clients are simply called "gateways", and the Management Server is referred to as "central server", which also hosts a front-end Web service called "Network Dashboard".

In the following sections we describe in more details all the components.

[1] http://projectbismark.net

5.1.1 BISmark gateways

BISmark home gateways consist of off-the-shelf routers running a customized firmware which enables its special features. The gateway periodically sends and heartbeat UDP packet to the central server, to allow the monitor of its state. Exploiting this packet the central server is also able to communicate back to the home router through eventual NATs, in order to update network configurations, to install software updates or to operate troubleshooting actions remotely.

We give more details about the currently adopted home routers in the following sections.

Selected hardware

The first version of the BISmark gateway was based on the Linksys WRT54GL router [113], which represents one of the most common and cheap wireless routers in the United States. It is equipped with a 200 MHz Broadcom BCM5352 processor, 16 MB of RAM, and 4 MB of flash memory, as well as five Fast Ethernet ports (one for WAN and four for LAN) and a 802.11b/g Wi-Fi adapter provided with two external antennas. Its original version mounts a proprietary Linux-based firmware which was replaced with a customized OpenWrt 8.09.2 distribution (codename Kamikaze).

Since working with significantly constrained resources does not allow to operate all kind of measurements, as detailed in Sec. 5.3, we decided to move to a different platform.

The second version of the BISmark home router was then based on the NOX Box [114], a small-form-factor computer resembling an off-the-shelf home router/gateway. It is based on an ALIX 2D13 6-inch by 6-inch single board computer equipped with a 500 MHz AMD Geode processor, 256 MB of RAM and at least 2 GB of Compact Flash II memory, as well as three Fast Ethernet ports and a Mini PCI 802.11b/g Wi-Fi adapter provided with two external antennas. The NOX Box natively runs a standard Debian Linux distribution which has been customized to avoid writing frequently on the flash memory for common operations (e.g. system log files). We worked on such platform keeping the same Linux distribution and porting to it the BISmark packages.

Although the NOX Box allows to perform any kind of experiment, enabling us to implement also passive measurements on a small controlled deployment, it resulted to be too expensive (i.e. about 300 US dollars) with respect to the networking equipment

provided. Indeed, most home users would have been happier to replace their existing router with a high-end gateway equipped with latest available technologies (e.g. 802.11n Wi-Fi adapter).

The latest version of BISmark now runs on top of a customized development version of OpenWRT (codename Backfire), which is supported by a variety of hardware platforms. We are currently using NetGear WNDR 3700v2 routers [115], which is equipped with a 680 MHz mips processor, 64 MB of RAM, and 16 MB of flash storage, as well as five gigabit Ethernet ports (one for WAN and four for LAN) and two Wi-Fi adapters (802.11 b/g/n and 802.11 a/n) respectively working on the 2.4 GHz and 5 GHz frequency bands.

Working with the latest router we were able to reach the best trade-off between price (i.e. about 130 US dollars) and equipment quality. Indeed, by exploiting part of the funds we received from Google, Intel, and NSF, we have been able to buy, configure, and deploy much more routers. As evidence of that, our sign-up model, where users can request a router signing a form on the project website, has generated over 20 K sign-ups since we launched it on May 18, 2011, which proves how the opportunity of taking home a top-of-the-line wireless-n router is enough incentive to join the project.

The BISmark firmware

As reported in the previous section, the latest version of the BISmark firmware is based on a branch we made from the official OpenWrt development trunk. We worked on our firmware in collaboration with the Bufferbloat project [116], by contributing also in developing and testing their CeroWRT distribution, which is aimed at investigating the problems of latency under load, bufferbloat, wireless-n, QoS, and the effects of various TCP algorithms on shared networks.

Besides the common functionalities provided by most wireless routers, the BISmark firmware comes with a set of packages implementing management and measurement functionalities, as required by the architecture design. The firmware currently includes a few main packages:

- `bismark-mgmt`: includes all the scripts and configuration files implementing the lightweight management protocol, which is based on UDP "heartbeat" packets periodically sent to the central server and allows it to communicate with the home router;

- `bismark-active`: includes all the configuration files and wrapper scripts enabling the router to properly manage underlying pre-existing active measurement tools;
- `bismark-chrome`: includes all the Web pages composing the default Web interface of the router, which provides BISmark users with handy documentation about the project, and links to the router configuration interface (based on Luci) and to the Network Dashboard.

More details on the communication protocols are discussed in Sec. 5.3. In addition to the main packages, the firmware also includes a small set of well-known active measurement tools:

- *D-ITG* [107]: platform capable of generating traffic at packet level accurately replicating appropriate stochastic processes for both IDT (Inter Departure Time) and PS (Packet Size) random variables (exponential, uniform, cauchy, normal, pareto, ...), which also acts as a tool for measuring most network performance metrics (e.g. one-way delay, round-trip latency, jitter, packet loss, throughput, ...) by using different transport layers (e.g. TCP, UDP, DCCP, SCTP, ...);
- *Netperf* [117]: benchmarking tool useful to measure the performance of many different types of networks, providing tests for throughput and end-to-end latency by using different transport protocols (e.g. TCP, UDP, SCTP, ...);
- *Shaperprobe* [6]: tool that non-intrusively probes network paths and tries to diagnose the presence of traffic shaping policies;
- *fping* [118]: tool utilizing the ICMP protocol to send echo requests to any number of hosts in parallel, sending out packets to hosts specified on the command line in a round-robin fashion.

The wrapper scripts provided by `bismark-active` package have the main purpose of running the previously listed tools feeding them with specific options, as specified in a configuration file, and to process their outputs in order to extract relevant statistics, which are then sent to the central server in a purposely defined XML format.

Additional Features

Besides the better performing hardware and the support for management and measurements, the BISmark gateway also provides many features that are not available on off-the-shelf routers:

- *Traffic shaping*: the routers (optionally) enforce various forms for traffic shaping, implementing HSFC/SFQ/RED93 based qos script, with SFB available as an option;
- *Bufferbloat mitigation*: the routers' firmware has been extensively analyzed and modified to compensate for and/or mitigate many potential 'bufferbloat' issues;
- *Additional diagnostic tools*: the routers are equipped with diagnostic tools for end users not commonly found in off-the-shelf home routers, including tcpdump, traceroute, and bing;
- *Caching Web proxy*: the *polipo* caching Web proxy is also installed, but not enabled by default.
- *Extensibility through advanced package management*: additional packages can be installed to enable PPPoE, VPN, MTR, etc.

We expect that many of the features enabled by our firmware enables, such as traffic shaping and bufferbloat mitigation, will allow BISmark users to experience better performance from their home networks than they currently enjoy with their existing home router equipment and configuration.

5.1.2 The central server

The BISmark central server represents the single point of control of the overall platform and is responsible for many tasks:

- It continuously monitors gateways availability by passively listening to "heartbeat" packets and, when a router is offline for more than a configurable time period, it sends a mail notification to the project maintainers;
- It allows the maintainers to remotely configure and update router firmware;
- Exploiting the possibility to enable SSH tunnels with the gateways, it allows the maintainers to remotely access their console to perform on-demand tests and troubleshooting.

- It is in charge of collecting measurement requests, assigning them to a specific measurement server, and avoiding their collision by means of a purposely designed scheduling algorithm;
- It acts as central repository to store measurement results, which are received by gateways in XML format and parsed to populate a PostgreSQL database.

We describe in the following the more relevant aspects of the central server.

The management daemon

In order to enable the remote management of gateways, the central server hosts a daemon called `bdmd` (BISmark Devices Management Daemon), which implements a multi-threaded UDP server and has been developed in C language. The main task assigned to the daemon is to listen for periodic "heartbeat" probes generated by the BISmark gateways. Such small UDP packets are the basis of the communication protocol, which we designed to allow the exchange of short messages between the central server and the gateways also in presence of NATs.

The protocol requires that every message exchange is always initiated by the home router and comprises the following ASCII encoded messages:

- `Client` → `Server`: all the messages sent from BISmark gateways to the central server contain their unique identifier, which in the latest firmware version includes the MAC address of the WAN interface (e.g. `OWC43DC7B0ADD9`). The following messages are currently supported:
  - `ping`: it is periodically sent to notify the router availability and its firmware version and to obtain from the central server the current public IP address of the gateway;
  - `measure`: it is sent to request the scheduling of a measurement and includes information about its name and expected duration, as well as the geographical zone (e.g. NorthAm) and the category of the requested measurement server[2]. The expected reply contains the IP address of the measurement server, possible options specific for each measurement (e.g. the port number to use), and a number representing the time to wait before starting the experiment;

[2]Measurement servers can be divided in groups depending on any useful criterion.

  - `log`: it is sent as a reply to specific commands (e.g. `config`), in order to give a feedback on their effect. It includes the command name, a timestamp, and the raw output generated by the specific command;

- `Server` → `Client`

  - `pong`: it is sent as a reply to the `ping command`, in order to establish the possibility to reach the central server. It includes the public IP address of the gateway and the server current timestamp, which is used to synchronize clocks at a coarse-grained precision as an alternative to NTP[3].
  - `fwd`: it can be solicited by administrators to request the creation of an SSH tunnel between the central server and a specific gateway; the tunnel can be then accessed to directly communicate with the gateway using any supported service (e.g. remotely accessing its Web interface).
  - `update`: it can be solicited by administrators to force the router to update its firmware, which is downloaded from the official repository using the HTTPS protocol, in order to guarantee the authenticity of the new firmware. A log reporting the success (or failure) of the update procedure is then sent to the central server using a `log` command;
  - `config`: it can be solicited by administrators to query or modify the current configuration of BISmark packages. A log reporting the all the current values of configuration variables is then sent to the central server using a `log` command;

To allow control messages to be sent asynchronously, `bdmd` temporarily stores them on a queue, implemented as a table on a SQLite database, and delivers them as soon as it receives a probe packet from the destination. Likewise, replies received from gateways are stored on the same table waiting to be read by administrators. Such mechanism allows to send command to gateway also when they are offline and assures the delivery of commands as soon as they are online again. This functionality is really useful when routers due to some unexpected issue, show up rarely and only short time windows are available to contact them to operate troubleshooting actions.

Different SQLite tables are used respectively to maintain the current status of all the deployed routers and to store the information needed to manage the scheduling of

[3]In some cases NTP protocol is filtered by restrictive firewall policices.

measurements. In details, for each measurement server we maintain its measurement capabilities, which include parameters specific to measurement services.

The management command-line interface

The BISmark central server provides a flexible command-line interface called `bdm` (BISmark Devices Manager). It enables administrators to remotely monitor, configure, update and control all the gateways.

```
bismark@dp4:~$ bdm list
Latest version: 462

Devices:
ID                IP               VERSION LAST_PROBE
NB105             74.176.79.18     302     2011-11-25 12:20:14
OWA021B7A9BF83    70.179.8.84      462     2011-11-25 16:31:53
OWA021B7A9BF95    67.186.227.63    462     2011-11-25 16:31:57
OWA021B7A9C409    75.68.52.189     462     2011-11-25 16:31:52
OWA021B7A9C655    76.111.39.202    462     2011-11-25 16:33:00
OWA021B7A9C85C    99.151.0.68      462     2011-11-25 16:32:36
OWA021B7AC738A    173.13.170.62    340     2011-11-25 16:32:37

Tunnels:
ID                PORT  START_TIME
OWC43DC79B5D25    40605 2011-10-28 19:11:53

Pending Messages:
ID      FROM    TO              MESSAGE
1       BDM     OWC43DC7A3EE43  fwd 35380
```

Figure 5.1: List of available gateways reported by `bdm`.

All the commands supported require as an argument the gateway unique identifier. We report here a brief description of the main commands:

- `config`: shows or changes BISmark configuration variables on the gateway;
- `upgrade`: upgrades the firmware of the specified gateway;
- `tunnel`: requests to the specified gateway the creation/closure of a reverse tunnel by using SSH port forwarding;
- `console`: gives access to the console of the specified gateway by using SSH over a previously established tunnel;
- `exec`: executes a command on the console of the gateway by using SSH over a previously established tunnel;

- `copy`: transfers a file to the `/tmp` directory on the specified gateway by using `scp` over a previously established tunnel;
- `list`: lists all the gateways which have reported at least once to the central server, as well as established tunnels and pending messages (see Fig. 5.1). Each line of the gateways list shows: the ID of the router, the IP address from which it is reporting, the version number of the BISmark software that it is running, and the date when the management server last received a report from the device.
- `mslist`: lists all the properties and capabilities of measurement servers (see Fig. 5.2) along with the currently supported measurement services (see Fig. 5.3). As explained in Sec. 5.3.3, the `FREE_TS` timestamp is used by the scheduling algorithm. The `MUTEX` boolean variable indicates if a certain measurement should be conducted in mutual exclusion with other activities on the measurement server;
- `readmsg`: displays the content of pending messages sent by the gateways and removes them from the queue[4];

Among the previous commands, `upgrade`, `tunnel` and `config` also allow to specify "ALL" as destination, which causes the delivery of the command to all the gateways.

```
bismark@dp4:~$ bdm mslist
Measurement Servers:
IP              ZONE     FREE_TS     SERVICES
143.215.131.173 NorthAm  1322240821  (HTTPDL,:8080/download.php) (HTTPUL,:8080/upload.php) (PING,0)
                                     (ITGDL,1430) (ITGUL,1430) (NETPERF,0) (RTR,1100) (SP,0) (TR,0)

143.225.229.126 Europe   1298564306  (HTTPDL,:8080/download.php) (HTTPUL,:8080/upload.php) (TR,0)
                                     (ITGDL,1430) (ITGUL,1430) (NETPERF,0) (PING,0) (RTR,110)
```

Figure 5.2: List of measurement servers capabilities returned by `bdm mslist`.

Data collection and storage

All the measurements conducted by BISmark gateways, produce results in the same purposely designed output format encoded in XML format. On every measurement cycle (by default 5 minutes) the router performs a dynamic set of experiments, whose composition depends on the different intervals associated to each measurement type. The execution of

[4]The messages are also stored on log file for troubleshooting.

the experiments set creates a local XML file (stored on volatile memory) which is then uploaded to the central server. On the other hand, the central server periodically scans the folder in which all the measurements are uploaded and parses the XML files to populate a PostgreSQL database with measurement data. Having such data stored on a DBMS was useful for many reasons. For instance, it allowed us to easily conduct complex analyses on a big amount of data. Moreover, it also feeds the Network Dashboard front-end interface with data necessary for plotting measurement results.

As shown in Fig. 5.4, each measurement comes with some meta-data and a fixed set of statistics. Most wrapper scripts produce one or more <measurement> tags each referring to a specific measured parameter. Having such standard reporting format allows us to

```
Measure Service Types:
TYPE        MUTEX
HTTPDL      yes
HTTPUL      yes
ITGDL       yes
ITGUL       yes
NETPERF     yes
PING        no
RTR         no
SP          yes
TCPDL       yes
TCPUL       yes
TR          no
```

Figure 5.3: List of measurement typologies reported by `bdm mslist`.

```
<?xml version="3.0" encoding="UTF-8" standalone="yes"?>
<measurements version="1.3">
  <info deviceid="OWA021B7A9C409" />
  <traceroute srcip="143.215.131.173" dstip="64.57.21.73" timestamp="1322244741" hops="13">
    <hop id="1" ip="143.215.131.1" rtt="1.288" />
    <hop id="2" ip="130.207.251.1" rtt="1.454" />
    <hop id="3" ip="130.207.254.45" rtt="1.437" />
    <hop id="4" ip="130.207.254.185" rtt="1.462" />
    <hop id="5" ip="143.215.194.85" rtt="1.865" />
    <hop id="6" ip="64.57.21.73" rtt="28.316" />
  </traceroute>
  <measurement param="RTT" tool="FPING" srcip="75.68.52.189" dstip="128.48.110.150"
               timestamp="1322244750" avg="115.600000" std="0.516398"   min="115.000000"
                                      max="116.000000" med="116.000000" iqr="1.000000" />
  <measurement param="LMRTT" tool="PING" srcip="75.68.52.189" dstip="73.194.80.1"
               timestamp="1322244744" avg="11.283000" std="3.994487" min="7.490000"
                                      max="19.226000" med="9.537500" iqr="3.742500" />
  <measurement param="JITTER" tool="DITG" srcip="143.215.131.173" dstip="75.68.52.189"
               timestamp="1322244956" avg="0.000340" std="0.000086" min="0.000191"
                                      max="0.000438" med="0.000367" iqr="0.000117" />
  <measurement param="PKTLOSS" tool="DITG" srcip="143.215.131.173" dstip="75.68.52.189"
               timestamp="1322244956" avg="0.000000" std="0.000000" min="0.000000"
                                      max="0.000000" med="0.000000" iqr="0.000000" />
</measurements>
```

Figure 5.4: Example of measurement results in XML format.

Figure 5.5: *Network Dashboard query interface*. The query interface shows the user a summary of the ISP service plan associated with the device, as well as the expected upload and download throughput rates for the service plan. The user can specify the performance parameters he wants to display in the plots, having also the possibility to compare the results with the median values of the same ISP or location. The "Passive Measurements" are only enabled on gateways where users gave their explicit permission and currently show the transferred bytes over time (cumulative and aggregated by well-known port numbers).

manage the results produced by very different tools with a unique set of utilities.

The front-end Interface: Network Dashboard

We have designed and implemented a front-end user interface for BISmark , the *Network Dashboard*, that allows the users to observe the network performance data that is measured and collected from their own home networks and to compare the performance that they receive to users with similar ISPs service plans. This was implemented in collaboration with Abhishek Jain and Nick Feamster at the Georgia Institute of Technology. The Network Dashboard allows users to answer questions such as:

- Is my ISP offering consistent performance over time?
- How does my performance compare to users with similar service plans?

We now describe the current capabilities of the Network Dashboard, in terms of the information that it provides to the users.

A user can plot various performance parameters using the interface shown in Fig. 5.5. Using this interface, the user can specify a time interval and the parameter to plot. The current interface allows a user to plot upload and download throughput, end-to-end and last-mile latency, and some basics passive measurements of traffic load. The design of the interface allows for the easy addition of plots of other measurements (e.g. packet loss, jitter).

(a) Download throughput (b) End-to-end Latencies

Figure 5.6: *Interactive performance plots.* This figure shows an example of the interactive plot interface provided by Network Dashboard, where performance parameters are reported for a given user over the course of a week. (a) Shows the download throughput measured with both single- and multi-threaded TCP transfers. (b) Shows the end-to-end latencies measured towards three different servers. The interface allows the user to see whether an ISP is providing the promised service over time. The interface also allows a user to change the observed time interval.

We report in the following two example to better illustrate the features offered by the Network Dashboard interface. On one side, Fig. 5.6(a) shows the download throughput rates that a particular user achieves from his upstream ISP over one week measured both with single- and multi-threaded TCP transfers. On the other side, Fig. 5.6(b) shows the "end-to-end latency" that a particular user achieves towards three different servers respectively located in California, in Georgia, and in Italy. For allowing the user to select a particular date range, the dashboard interface provides some buttons to switch between preconfigured time intervals ranging from one day to one year or to show all the data available from the first to the last measure ever. Hovering the pointer over a particular portion of the graph also shows the absolute values for a particular point. In addition to time-series plots, the Web interface shows also the upstream and downstream throughput rates that the user can *expect* from the service plan, to allow him to compare expected and achieved rates.

Download of Raw Data Files The central server, after parsing XML files, keeps a gzip compressed archive of them, which may be useful to properly re-populate database tables if some bug is detected in parsing scripts. The Network Dashbord, upon explicit request, allows users to download these XML files, which are accessible through a password protected page reachable from a "Download XML" link (see. Fig. 5.5). In addition to the metrics available from the plots, the raw data files also include other active measurements,

as listed in Table 5.1. By design these other active measurements can be easily integrated into the graphical front-end, as well.

5.1.3 Measurement servers

As specified by the architecture proposed in Sec.3.2, the measurement servers consist in high-end servers provided with high capacity access to the Internet. Their role is very simple: host the sender and receiver side of cooperative active measurement tools. The measurement server is provided with wrapper scripts (each tool has a dedicated wrapper) to properly feed tools with the needed command-line options. In case a certain measurement tool does not provide the possibility to enable the initiation of a measurement from the server side, by using the `socat` utility we implemented a listener on a specific port number which is responsible for launching the wrapper script and to pass it the needed parameters, which are received over the TCP socket from the gateway. For instance, by exploiting such mechanism, we implemented a reverse traceroute measurement initiated upon request by the gateway. In addition, in order to cope with unexpected problems generated by underlying tools, the server periodically checks system resources and kills and reloads measurement daemons in case of unusual resource consumption or stalled processes.

Beyond that, the measurement servers do not need to directly communicate with the central server. Indeed, the scheduling algorithm executed by the central server does not need to communicate with them. This potentially allows to manage also measurements conducted towards server not under our control.

Currently the BISmark platform directly hosts two dedicated measurement servers, one at the University of Napoli and the other at Georgia Institute of Technology. Besides those servers, the platform also relies on properly configured servers belonging to the MLAB open platform [112], which counts many servers spread all around the world.

5.2 Measurements

BISmark gateways are capable of conduction both active and passive measurements. Most gateways deployed only perform active measurements for two major reasons:

- Collecting passive measurements involves privacy issues, thus requiring a formal and explicit permission by users;

- The adopted hardware platform does not provide enough resources to perform passive measurements.

We give in the following sections about the measurements currently supported by our gateways.

5.2.1 Active measurements

The BISmark gateways periodically perform each active measurement at the frequency reported in Table 5.1, regardless of whether there is cross traffic on the link. In addition, they also allow to perform on-demand measurements by exploiting the remote management mechanism and to easily add new measurement techniques and tools.

The gateway currently collects measurements relating to throughput, latency, packet loss, and jitter, as well as DNS delay and failures, and routes.

BISmark measures *bulk transfer capacity* by performing both single- and multi-threaded TCP downloads and uploads using Netperf [117] for 15 seconds once every 30 minutes. To account for cross-traffic, we count bytes transferred by reading directly from `/proc/net/dev`, and compute the *passive throughput* as the byte count after the TCP transfer minus the byte count before the transfer, divided by the transfer time. This gives us the combined throughput of the TCP transfer and the cross traffic.

We also measure *end-to-end capacity* using ShaperProbe [119] once every twelve hours. All measurements are synchronized to avoid overlapping of invasive experiments towards the same measurement server, as will be explained in Sec. 5.3.3.

BISmark gateways measure *end-to-end latency* to a nearby wide-area host, and two additional latency metrics we found to be critical:

- *last-mile latency*: the end-to-end latency to the first public IP hop towards the Internet;

- *last-mile latency-under-load*: the last-mile latency when the upstream link is saturated.

It also measures *packet loss* and *jitter*, by generating low-rate UDP packet trains using the D-ITG tool [107], *DNS latency* and *DNS failures*, by resolving top-10 domain names using the well-known nslookup tool, and *forward and reverse routes*, by initiating standard traceroute measurements from both the gateway and the selected measurement server.

Parameter	Type	Prot.	Freq.	Comments
BISmark : 17 devices, 3 ISPs				
Latency	End-to-end	ICMP	5 min	Host
	Last-mile	ICMP	5 min	First IP hop
	Upstream load	ICMP	30 min	During upload
	Downstream load	ICMP	30 min	During download
Packet loss	End-to-end	UDP	15 min	D-ITG
Jitter	End-to-end	UDP	15 min	D-ITG
Downstream Throughput	Single-thread HTTP	TCP	30 min	curlget to Host
	Passive throughput	N/A	30 min	/proc/net/dev
	Capacity	TCP	12 hrs	D-ITG
	Capacity	UDP	12 hrs	ShaperProbe
Upstream Throughput	Single-thread HTTP	TCP	30 min	curlput to Host
	Passive throughput	N/A	30 min	/proc/net/dev
	Capacity	TCP	12 hrs	D-ITG
	Capacity	UDP	12 hrs	ShaperProbe
DNS	Delay and Failure	UDP	15 min	Top-10 Alexa

Table 5.1: Active measurements periodically collected by BISmark .

For each of the above metrics we compute on-the-fly the following set of statistics: mean, standard deviation, minimum, maximum, median and inter-quartile range. Together with some meta-data (e.g. timestamp, source and destination IP addresses, used tool, ...), these statistics are sent to the central server as explained in Sec. 5.1.2.

5.2.2 Passive measurements

Pursuing a collaboration in the context of the HNDR (Home Network Data Recorder) project [120], we added a set of passive measurements to a small subset of BISmark gateways, which were deployed in the city of Atlanta from June to December 2010. In order to do that we have been forced to switch to the Nox Box platform, because the resources provided by the Linksys WRT54GL were not sufficient. To cope with privacy issues, all the data collected is anonymized before sending it to the central server and all the users involved are formally asked for permission.

Thanks to the generous availability of memory offered by the Nox Box (see Sec. 5.1.1), we have been able to implement many different passive measurements.

We collected flow level data (capturing on all available network interfaces) using the TIE platform [87], which is able to associate each flow with a label representing the application guessed as a function of both the transport-layer port number and the first packets' initial payload bytes, by adopting the *portload* technique [121]. TIE has also been extended to extract statistics at packet-level (i.e. packet size, inter-packet time) and related to HTTP requests. We also collected statistics about Wi-Fi at data-link level (e.g. signal strength and quality, number of transmitted data and beacon packets, ...)

using the airodump-ng tool [122], which was configured to listen by default only on the same channel as the managed Wi-Fi home network. In addition, we monitored ARP associations and all the events associated to DHCP protocol and gateway, modem and uplink availability.

By performing this small-scale deployment we were able to evaluate the efficiency of the selected measurements in characterizing the network usage profile of home users. Moreover, we found that collecting flow-level and data-link level statistics produces a huge amount of data, which makes it difficult to guarantee acceptable delays when querying the related database tables.

5.3 Challenges and solutions

While designing and implementing the platform we had to face some challenges, mostly caused by the limited resources offered by the gateways. In the following sections we describe the solutions we adopted to solve such issues.

5.3.1 Hardware constraints

Working with the Linksys WRT54GL router, initially chosen due to its wide diffusion and low price, we encountered many difficulties. Indeed, providing only 4 MB of flash and 16 MB of RAM memory, it was challenging to manage the coexistence of the operating system, the measurement tools and the measurement results. In order to solve this problem we adopted the following solution:

- we built all the binaries part of the BISmark packages by applying the highest level of optimizations and by stripping unnecessary symbols.
- we stored on flash memory only the smaller files, while the others were downloaded at run-time, during the bootstrap process, and stored on ram-disk;
- we implemented all the control and communication logic in *ash* scripting language, whose interpreter is a basic component of the OpenWRT operating system and thus is already included;
- we organized all the measurements to process their output on-the-fly, by exploiting the *awk* interpreter and the pipelining mechanism, so that the final output is directly

encoded in XML format and temporarily stored on ram-disk, thus allowing us to avoid the storage of intermediate results;

- we forced the results to be uploaded as soon as possible to free the space occupied space.

All the previous optimizations allowed us to reduce the need for storage to the essential, thus gaining enough resources to run all the active measurements listed in Tab. 5.1.

5.3.2 Lightweight reliable remote management

The BISmark gateway was designed aiming at offering both the normal functionalities of a common home router and the measurements necessary to evaluate the performance of the access link. Since they are deployed in real houses, where real users just want to obtain good performance when using their favorite applications, they should try to impact as less as possible on user perceived performance. Therefore, to pursue this objective, the traffic generated by both management and measurement activities should be reduced as much as possible.

Depending on the policies adopted by the ISP, some constraints may prevent the communications between the gateway and the central server.

On the one hand, some ISPs assign to home routers private IP addresses, thus making it impossible the central server to initiate a communication with them. In addition, the BISmark router in some cases may be deployed behind the router originally provided by the ISP which implements NAT. In such cases the BISmark gateway may result to be behind multiple NAT levels. Aside from this case, the gateway itself is configured by default to deny the access to the SSH console from the WAN interface.

On the other end ISPs may apply restrictive outbound filtering rules, which allow only a small set of destination port numbers to be contacted from the gateway. Hence a reliable remote management control protocol should be designed to work properly in all these conditions.

The management protocol we designed, as described in Sec. 5.1.2, is the result of such guidelines. During normal operations a gateway only generates small UDP packets at random intervals between 60 and 90 seconds. The variability of such interval allows to avoid the accidental synchronization between different routers, which may result in burtst of UDP packets received by the central server. Despite the simplicity of such UDP packet,

it enables the implementation of a reliable remote management protocol. Indeed, being it generated by the gateway, the packet enables the central server to reply with another UDP packet bypassing any number of NATs. Moreover, to overcome outbound filtering rules, we configured the `bdmd` daemon to listen on multiple port numbers, which are tried by the gateway with a round-robin fashion.

Once such UDP packet is able to reach the central server it allows to create on demand a reverse SSH tunnel, which can be used to remotely administer the gateway over a secure and reliable communication channel.

To be completely sure to be able to communicate with gateways once they are deployed far away and all the previous mechanisms fail for some reason, we provided them with a recovery tunnel mechanism which automatically creates the reverse tunnel if the gateway is unable to contact the server for more than 20 minutes.

5.3.3 Measurements collision

The BISmark platform, according to the general properties of the architecture defined in Sec. 3.2, provides the support form a big number of gateways and multiple measurement servers, whose number is much smaller. In these conditions arises the problem of avoiding collisions between measurements conducted by different gateways towards the same measurement server. In addition, since we are interested in measuring the performance of the access link, each gateway should conduct measurements towards the closest available measurement server, in order to reduce the effect on the results by additional links on the path.

Following these guidelines we designed a centralized scheduling algorithm, managed by the central server, which works with two different class of measurements: LIGHT and INVASIVE. It is based on a timestamp (called FREE_TS) associated to each server and completely avoids the overlap of INVASIVE measurements towards the same measurement server. The FREE_TS value represents the instant in which the measurement server will be available to conduct an INVASIVE measurement. Whenever a measurement request is received along with its duration, the central server selects among the measurement servers in the same geographical area the one having the smallest FREE_TS value. If such value is in the past with respect to the current time, the measurement is immediately started and the FREE_TS value is incremented by the duration declared for the experiment plus 5 additional seconds, which allow to avoid border effects between consecutive measurements.

On the other hand, if the FREE_TS value is in the future, the measurement is queued on the selected server by simply incrementing FREE_TS as in the previous case, but the reply sent back to the gateway forces it to wait for a time period equivalent to the difference between FREE_TS and the current time.

5.4 A study of broadband in the USA from the gateway

In the following sections we briefly describe the first BISmark deployment, which allowed us to conduct an analysis of home access networks in the city of Atlanta, by providing a detailed view of performance in the context of a broader study conducted in collaboration with SamKnows [123]. Hence, we report important lessons learnt from this study about throughput and latency in presence of modem buffering and ISPs traffic shaping policies.

The first BISmark deployment (see Fig. 5.2), which we ran in January 2011, had 14 gateways across AT&T (DSL) and Comcast (Cable), with 2 more on Clear (WiMax). Among them, the AT&T users form the most diverse set of users in the deployment, with five distinct service plans.

Our goal with the measurements from this deployment was to achieve *depth*: the BISmark platform allowed us to take measurements with detailed knowledge of how every gateway is deployed; we could also take repeated measurements and conduct specific experiments on-demand, imposing different settings and configurations.

5.4.1 Understanding Throughput

In this section we explore how the different throughput measuring techniques in BISmark generate different results and how to interpret them. We also explore the traffic shaping in the Comcast network.

Why Do We Use Different Throughput Measurement Techniques?

Different throughput measurement techniques measure different aspects of throughput. We compare several methods for measuring throughput from Table 5.1. We normalize each throughput measurements to the service plan rates advertised by the ISP so that we can compare performance across access links where users have different service plans.

ISP	**Technology**	Total Deployments
Comcast	Cable	4
AT&T	DSL/FTTN	10
Clear	Wimax	2

Table 5.2: The first BISmark deployment.

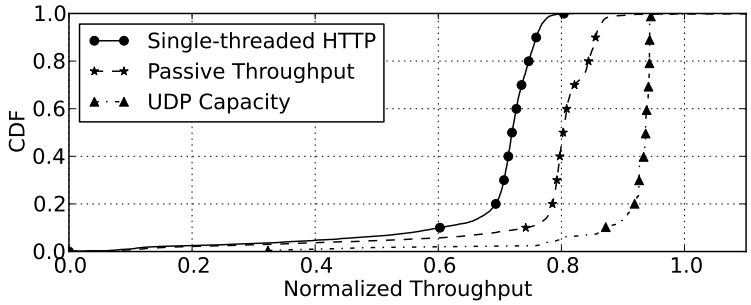


Figure 5.7: Comparison of various methods of measuring throughput.

Figure 5.7 shows the normalized throughput reported by the three methods we presented in Table 5.1 for two AT&T users. The plot is a CDF, and the x-axis denotes the normalized measurement. As we can see, the three techniques show different measurements. UDP measurements obtained from ShaperProbe produce consistent measurements of throughput that are closest to the service plan. Single-threaded TCP does not achieve the same throughput, most likely due to the inefficiencies associated with TCP, and also potentially protocol overhead and loss on the access link. This technique could also be affected by cross traffic generated by users in the network. Passive measurements provides a better estimate of the throughput as it accounts for cross traffic and higher-layer protocol overhead. What this shows is that the capacity of the access link is fairly consistent, but the ability of applications to use it is less so.

Effect of Traffic Shaping on Throughput

As explained in Sec. 1.1.1, *PowerBoost*^{TM} is a technology that allows users to experience throughput higher than their service plan for short periods of time, which is usually applied only in downstream. Comcast [124], explains that *PowerBoost*^{TM} provides higher throughput for the first 10 MBytes of a download and the first 5 MBytes of an upload. We measure the shaped throughput for download and upload at the receiver using `tcpdump`. Because our tests are intrusive, we conducted them only a few times; however the results do not vary with choice of traffic generators or ports.

We use BISmark to study the implementation of *PowerBoost*^{TM} on Comcast access links. Figure 5.8 shows the observed download throughput for four users. All of them see *PowerBoost*^{TM} effects, but, surprisingly, we see many different profiles even in such a small subset of users. Figure 5.8 shows download profiles for each user (identified by the

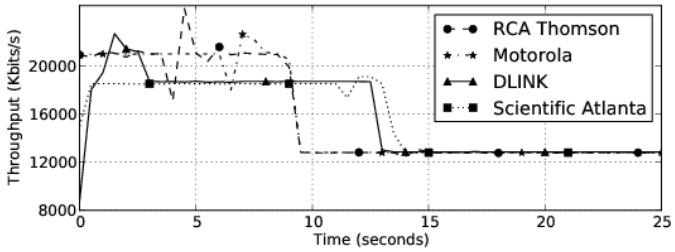


Figure 5.8: *PowerBoost*^{TM} download behavior for 4 users.

modem they use; while the modem does not have an effect on burst rates, it does have an effect on buffering latencies as we show in Sec. 5.4.2). The user with a D-LINK modem sees a peak rate of about 21 Mbps for 3 seconds, 18.5 Mbps for a further ten seconds, and a steady-state rate of 12.5 Mbps. The Motorola user sees a peak rate of 21 Mbps for about 8 seconds. The *PowerBoost*^{TM} technology provides token buckets working on both packet and data rates [125]; it also allows for dynamic bucket sizes. The D-LINK profile can be modeled as a cascaded filter with rates of 18.5 Mbps and 12.5 Mbps, and buffer sizes of 10 MBytes and 1 MByte respectively, with the line capacity being 21 Mbps. Because our results do not vary with respect to the packet size, we conclude that Comcast does not currently apply buckets based on packet rates.

We repeated these experiments different traffic generators (`iperf` and D-ITG) and port numbers and witnessed the same effect in all cases. More important than the actual numbers is the fact that dynamic traffic shaping exists and that it varies widely even among a small subset of users. The different rates of the burst suggest that the implementation could be either a single token bucket in the case of a single burst, or cascaded buckets in the case of multiple burst sizes.

We also confirm a characteristic of token buckets, the dependence on volume. To confirm the token bucket assumption, we designed a new experiment by generating intermittent loads with a constant long term rates equal to the maximum burst detected and varying ON/OFF periods. This experiment confirms a property typical of token bucket filters: the dependency on the traffic volume. Figure 5.9 shows the effect of intermittent load (21.5 Mbps for 5 seconds, off for 2 seconds for download, 8 Mbps for 5 seconds, 2 seconds off for upload) on the user with the D-LINK modem. This traffic profile experiences burstiness for longer (nearly 23 seconds as opposed to about ten seconds, compared with Figure 5.8) as tokens replenish during the OFF period; therefore, the sender can

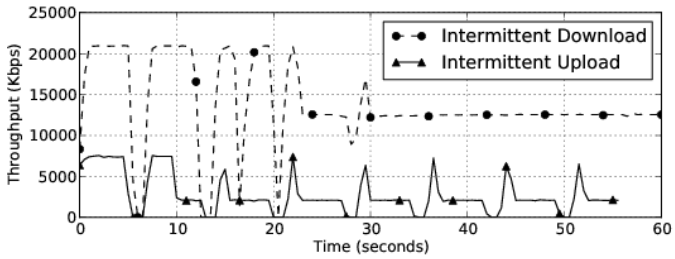


Figure 5.9: Generating intermittent traffic load confirms that the shaping mechanism is volume-based.

sustain peak rates for longer (although it should be noted that the volume of traffic sent at higher rates is the same as with continuous loads. By periodically switching ON and OFF the traffic load, the shaping effect is delayed proportionally to the generated traffic volume. For instance, the shown intermittent download rate is affected from shaping at 12.5 Mbps only after about 23 seconds, because the OFF periods let the bucket recover enough tokens to sustain more bursts.

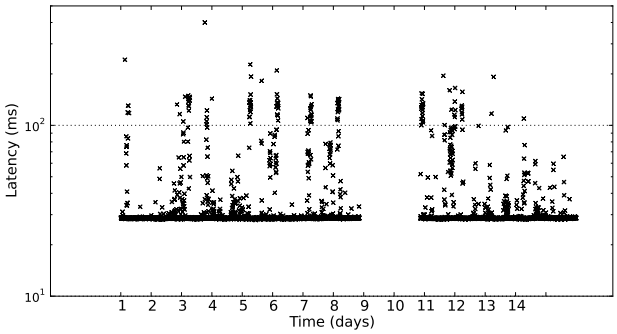
PowerBoost^{TM} also exhibits interesting latency behavior, which we explore in more detail in the following section 5.4.2.

5.4.2 Latency and Buffering

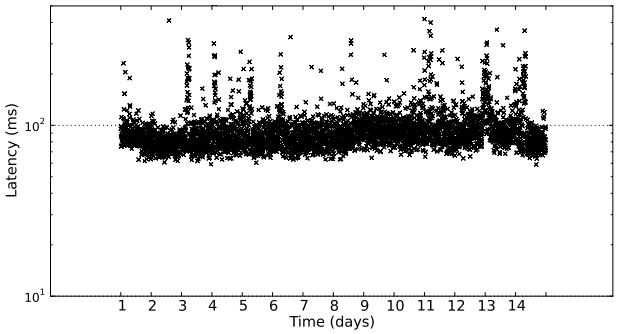
In this section we explore how latency is affected by access technologies, modem buffering, and ISP policies.

Fig. 5.10 shows the latency experienced over a two weeks period by three users, one each in Cable, DSL, and WiMax ISPs. It is clear at a glance, that wired access technologies obtain much stable latencies than wireless ones. Indeed, Comcast and AT&T users generally achieve low latency variance. On the other side, the two wired technologies observe different values for such metric, being Cable latency nearly one order of magnitude below the DSL one.

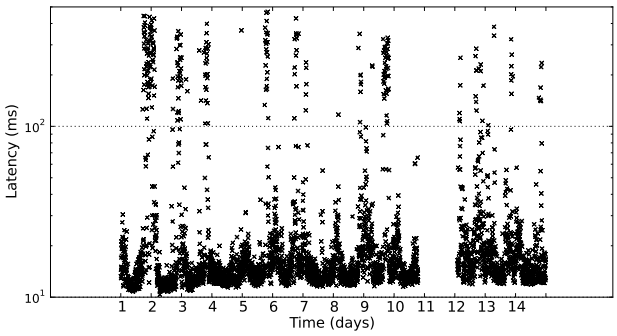
The Comcast user (see Fig. 5.10(a)), achieves most of the time a very stable latency of about 30 ms, with some sporadic spikes up to 400 ms. We discuss the origin of such effect in Sec. 5.4.2. The AT&T user (see Fig. 5.10(b)) despite the high baseline latency of 100 ms, which is due to interleaving, observes consistent latency values characterized by a random variability around the baseline value. Finally, the WiMax user (see Fig. 5.10(c)) presents a high variability in latency and is clearly affected by time-of-day effects, which cause latency to increase up to 500 ms during peak hours.



(a) Comcast user (Cable).



(b) AT&T user (DSL).



(c) Clear user (WiMax).

Figure 5.10: Latency profiles for different access technologies.

This simple analysis allows us to confirm that access technologies have a big impact on metrics like latency and jitter and they suffer from different issues depending on their weaknesses.

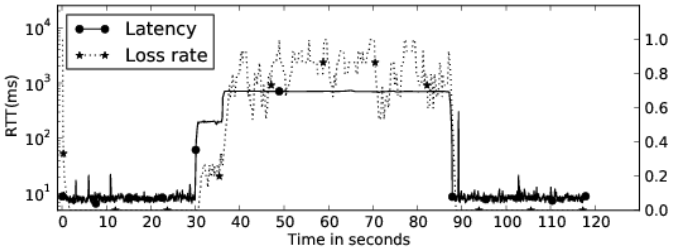


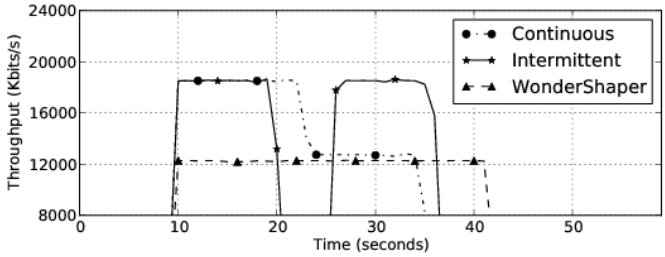
Figure 5.11: Comcast user with D-LINK modem.

Latency Under Load

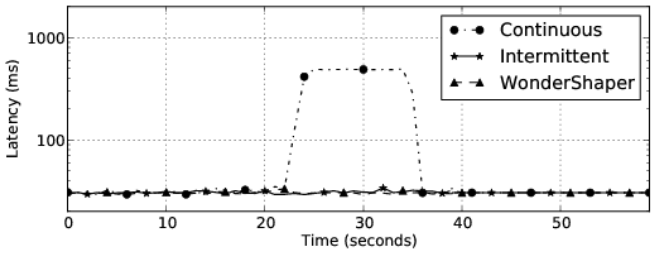
Buffers on DSL and cable modems they absorb bursty traffic and enable smooth outflow at the configured rate [8]. Buffering only affects latency during periods when the access link is loaded and packets can see substantial delays as they queue up in the buffer. The capacity of the uplink also affects the latency introduced by buffering. Given a fixed buffer size, queuing delay will be lower for access links with higher capacities because the draining rate for such buffers is higher. If the buffer is full, then packets will have more or less steady latency that is proportional to the size of the buffer. To study the effects of modem buffers on latency under load, we conduct tests on a user with the Comcast (12.5 Mbps down; 2 Mbps up) plan. We perform the following experiment: we start ICMP `ping` at the rate of 10 pkts/s to the last mile hop. After 30 seconds, we flood the uplink at 10 Mbps for Comcast using `iperf` UDP). After 60 seconds, we stop `iperf`, but let `ping` continue for another 30 seconds. The `ping` measurements 30 seconds on either side of the `iperf` test establishes baseline latency. All of the modems we study have buffers that induce less than one second of delay, but these users see surprising latency under load profiles due to traffic shaping. Figure 5.11 shows the latency under load for a Comcast user. This user sees a jump in latency when the flooding begins and about 8 seconds later, *another increase* in latency.

Packets see lower latencies during *PowerBoost*^{TM} because, for a fixed buffer, the latency is inversely proportional to the draining rate. The increase in latency due to *PowerBoost*^{TM} (from 200 ms to 700 ms) is proportional to the decrease in the draining rate at the end of *PowerBoost*^{TM} (from 7 Mbps to 2 Mbps for this user).

Can data transfer be modified to improve latency under load? We explore whether a user can modify their data transfer behavior so that large "bulk" flows and



(a) Throughput.



(b) Latency.

Figure 5.12: Maintaining low latency by modifying data transfer behavior.

delay-sensitive flows can co-exist without interfering with one another. For instance, such behavior could be useful if the user wants to play online games or to make VoIP calls while downloading big files. We compare the impact of a 50 MByte download on a G.711 VoIP call in three different conditions: (1) not applying any traffic control, (2) intermittent traffic at capacity on 10.8 seconds ON and 5.3 seconds OFF cycle, and (3) shaping using the WonderShaper [126] approach. Figure 5.12 shows the result of this experiment. In (1), the transfer takes 25.3 seconds; however, just after the *PowerBoost*^{TM} period, the VoIP call starts suffering high latency and loss until the end of the transfer. In (2), traffic is sent in pulses, and the download takes 26.9 seconds. In (3), traffic is sent at just under the long term rate and the download takes 32.2 seconds. Both (2) and (3) do not increase latency significantly, this is because they do not deplete the tokens at any time, and therefore cause no queuing. In approach (2), the ON/OFF periods can be configured depending on the token bucket parameters,[5] and the size of the file to be transferred. Both approaches achieve similar long-term rates but yield significant latency benefit. The drawback is that any approach that exploits this behavior would need to know the shaping parameters.

[5] If ρ_r is the rate we want to reserve for real-time applications, and ρ_t the token rate, the condition to be satisfied is: $(\rho_b + \rho_r - \rho_t) \times \tau_{on} \leq \tau_{off} \times (\rho_t - \rho_r)$, where ρ_b is the sending rate during the pulse, and τ_{on} and τ_{off} are the ON and the OFF times, respectively.

Chapter 6

Conclusion

The Internet today counts about 2.1 billion users worldwide, thus representing a highly complex system on which huge amounts of data are transferred every day over the IP protocol. Most people rely on Internet connectivity for everyday activities, making broadband access an essential resource, whose performance have not been widely studied in literature. On one hand, evaluating IP networks performance is a challenging task which is made even more difficult by the heterogeneity of underlying protocols and technologies. On the other hand, the complexity of emerging applications makes it difficult to understand the relation between network and user-perceived performance.

In this thesis we addressed the evaluation of IP networks performance with a specific focus on broadband access networks. We started presenting in Chapter 1 an overview of the Internet scenario with respect to the heterogeneity of access technologies and the complexity of new generation applications. Hence, we described in Chapter 2 the research activities we performed on both the analysis and characterization of traffic generated by new-generation applications and the identification of relevant metrics, methodologies, techniques and tools to evaluate network performance. We introduce in Chapter 3 our study on the evaluation of broadband access networks performance, in which we propose a taxonomy of existing approaches and define the guidelines to build an architecture with ideal characteristics to measure access networks performance on a large scale. Finally, we respectively described in Chapters 4 and 5 the host- and router-based architectures we designed, implemented and deployed, by detailing the challenges we faced and the solutions we adopted to solve them.

Our thesis includes several contributions to the field of IP networks performance evaluation.

By analyzing the traffic of new-generation applications, we obtained a better understanding of their characteristics and behavior. Such applications are progressively providing - through a single interface - more interactions among the users and between the users and the network. We define as *multi-channel applications* those providing a single interface to perform heterogeneous activities exploiting multiple communication channels. Very often such applications rely on secret proprietary protocols and their communications are heavily encrypted, thus making their study a challenging task. Hence, we proposed a novel methodology which allows to identify different activities performed by multi-channel applications, thus helping to characterize them with higher accuracy and to detect behaviors and aspects otherwise not visible.

By performing an extensive study of metrics for measuring network performance, we established that, among the many metrics defined in literature, their selection and interpretation has to be made depending on the specific scenario and on the considered network applications. Moreover, different methodologies, techniques, and tools can be adopted to measure such metrics and their results may be not easily comparable.

Since, thanks to the previous research activities, we identified a complex relation between the performance of the network and of applications, we found that most users are not able to properly understand low-level network metrics and how them affect their perceived performance. Moreover, ISPs usually advertise their broadband service plans only in terms of download and upload speeds, which is not sufficient to describe the performance that user can expect when using various applications. Hence, we proposed a novel approach to present performance metrics to the user in a standard format. In analogy with "Nutrition Labels" for food items, such format reports both low-level parameters - represented as ranges of possible values - and high-level concepts (e.g. time to download a song, video streaming quality in terms of frames rate, ...).

By focusing our research activities on the study of broadband access networks, we performed an extensive analysis of existing projects addressing their performance evaluation, which brought us to the definition of a taxonomy for broadband benchmarking approaches. We found that most projects adopt different techniques and methodologies and none of them consider a complete set of network performance metrics. Moreover, we identified router-, client-, and plugin-based approaches as the most promising, since they allow to consider the context in which measurements are conducted and to perform experiments on the same access link properly scheduled over a long time period. The

knowledge acquired in the previous activities, brought us to the definition of the ideal characteristics for an architecture specialized to evaluate the performance of broadband access networks on a large scale.

Following the guidelines previously defined, we designed and implemented two architectures respectively adopting the host- and router-based approaches.

On one hand, we adopted a client-based approach, which allows to easily reach large scale deployments and to evaluate the performance of broadband access networks at higher geographical resolution. We designed and implemented the HoBBIT platform, which is specifically targeted to evaluate the performance of broadband access networks in Italy.

During this activity we devised new methodologies and techniques to cope with several challenges arose during the different phases from the design to the deployment. For instance, we performed an extensive analysis of the scalability of the platform on both the management of network resources and on the database performance, in order to quantify its limitations and to devise the strategies necessary to support its growth. Hence, we conducted a preliminary analysis of some home access networks in Italy which brought to some interesting results.

On the other hand, we adopted a router-based approach, which allows to solve most of the issues encountered by all the other approaches, but makes it challenging to obtain a large scale deployment. We designed and implemented the BISmark platform, which is based on a customized firmware compatible with most off-the-shelf routers. During this activity we had to face some challenges, mostly caused by the limited resources offered by the gateways, for which we proposed innovative solutions. Hence, we conducted an analysis of home access networks in the city of Atlanta (i.e. 14 BISmark gateways), by providing a detailed view of performance in the context of a broader study conducted in collaboration with SamKnows [123]. While the SamKnows first-stage large scale deployment (i.e. 4000 gateways in the United States) provided *breadth* by classifying a large set of users across a diverse set of ISPs and geographical locations, with BISmark we were able to achieve *depth*: our platform allowed us to conduct measurements with detailed knowledge of how every gateway is deployed and to conduct repeated measurements and specific experiments with different settings and configurations. Therefore, we found that different methods of measuring throughput may produce different results. Moreover, different ISPs use different policies and traffic shaping behaviors that make it difficult to compare measurements across them. We also observed that different access technologies

have a different impact on latency. Finally, we detected that modem buffers can introduce significant latency variations that are orders of magnitude more than the variations introduced by the ISP.

Thanks to the proposed methodologies and to the realized platforms we enabled the scientific community to better address the study of performance in the context of broadband access networks. Indeed, our contributions can help research activities in scenarios different from broadband access networks. For instance, they can help in the characterization and analysis of enforced QoS policies (e.g. bandwidth throttling), network neutrality, Internet universal services, home networks and applications.

Bibliography

[1] Internet World Stats. http://www.internetworldstats.com/dsl.htm.

[2] 2010 global - key telecoms, mobile and broadband statistics. https://www.budde.com.au/Research/2010-Global-Key-Telecoms-Mobile-and-Broadband-Statistics.html.

[3] Netalyzr. http://netalyzr.icsi.berkeley.edu/.

[4] Matt Mathis et al. Network Path and Application Diagnosis. http://www.psc.edu/networking/projects/pathdiag/.

[5] Richard Carlson. Network Diagnostic Tool. http://e2epi.internet2.edu/ndt/.

[6] P. Kanuparthy and C. Dovrolis. Shaperprobe: End-to-end detection of isp traffic shaping using active methods. 2011.

[7] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing residential broadband networks. In *Proc. ACM SIGCOMM Internet Measurement Conference*, San Diego, CA, USA, October 2007.

[8] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *Proc. Internet Measurement Conference*, Melbourne, Australia, November 2010.

[9] G Maier, A Feldmann, V Paxson, and M Allman. On dominant characteristics of residential broadband internet traffic. In *ACM Internet Measurement Conference*, 2009.

[10] National Broadband Plan. http://www.broadband.gov/.

[11] Karl Bode. FCC: One Million Speedtests and Counting. http://www.dslreports.com/shownews/FCC-One-Million-Speedtests-And-Counting-109440, July 2010.

[12] Dave Vorhaus. A New Way to Measure Broadband in America. http://blog.broadband.gov/?entryId=359987, April 2010.

[13] Communications infrastructure report 2011. http://stakeholders.ofcom.org.uk/binaries/research/telecoms-research/bbspeeds2011/bb-speeds-may2011.pdf, 2001.

[14] Ulteriori disposizioni in materia di qualità e carte dei servizi di accesso ad internet da postazione fissa ad integrazione della delibera n.131/06/csp. Delibera N. 244/08/CSP, 2008.

[15] Eu broadband test. http://www.tp-link.com/en/article/?id=1074, 2011.

[16] TR-143 Amendment 1. Enabling network throughput performance tests and statistical monitoring. DSL Forum Technical Report, 2008.

[17] Asymmetric Digital Subscriber Line (ADSL) Transceivers. ITU-T G.992.1, 1999.

[18] Asymmetric Digital Subscriber Line (ADSL) Transceivers - Extended Bandwidth ADSL2 (ADSL2Plus). ITU-T G.992.5, 2003.

[19] Data-over-cable service interface specifications: Radio-frequency interface specification. ITU-T J.112, 2004.

[20] C. Bastian, T. Klieber, J. Livingood, J. Mills, and R. Woundy. *Comcast's protocol-agnostisc congestion management system*. Internet Engineering Task Force, December 2010. RFC 6057.

[21] T. Koonen. Fiber to the home/fiber to the premises: What, where, and when? *Proceedings of the IEEE*, 94(5):911 –934, may 2006.

[22] ANSI/IEEE Stdandard 802.11. http://standards.ieee.org/getieee802/download/802.11-2007.pdf as of November 2011.

[23] G. Xylomenos, G.C. Polyzos, P. Mahonen, and M. Saaranen. Tcp performance issues over wireless links. *Communications Magazine, IEEE*, 39(4):52–58, Apr 2001.

[24] Ieee 802.16m - advanced mobile broadband wireless standard. IEEE Standards Association. March 31, 2011.

[25] Emir Halepovic, Carey L. Williamson, and Majid Ghaderi. Wireless data traffic: a decade of change. *IEEE Network*, 23(2):20–26, 2009.

[26] J. Vinamaki. Ip over satellite. *Procedings of Innovaton Dinamic and Mobile Communication*, 2004.

[27] G.L. Fong and K. Nour. Broadband and the role of satellite services. http://www.frost.com/prod/servlet/cpo/11139757 as of November 2011. Frost and Sullivan.

[28] J. Bem, T. Wieckowski, and R. Zielinski. Broadband satellite systems. http://www.comsoc.org/livepubs/surveys/public/1q00issue/zielinski.html as of November 2011. Wroclaw University of Technology.

[29] Wei Li, Marco Canini, Andrew W. Moore, and Raffaele Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *Comput. Netw.*, 53:790–809, April 2009.

[30] F. Michaut and F. Lepage. Application-oriented network metrology: metrics and active measurement tools. *Communications Surveys Tutorials, IEEE*, 7(2):2 – 24, quarter 2005.

[31] TR-069 Amendment 2. Cpe wan management protocol. DSL Forum Technical Report, 2007.

[32] Cesar D. Guerrero and Miguel A. Labrador. Traceband: A fast, low overhead and accurate tool for available bandwidth estimation and monitoring. *Comput. Netw.*, 54:977–990, April 2010.

[33] Karthik Lakshminarayanan, Venkata N. Padmanabhan, and Jitendra Padhye. Bandwidth estimation in broadband access networks. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pages 314–321, New York, NY, USA, 2004. ACM.

[34] S. Sundaresan, N. Feamster, R. Teixeira, A. Tang, K. Edwards, R. Grinter, M. Chetty, and W. de Donato. Helping users shop for isps with internet nutrition labels. In *ACM SIGCOMM Workshop on Home Networks*, 2011.

[35] S. Bauer, D. Clark, and W. Lehr. Understanding broadband speed measurements. In *38th Research Conference on Communication, Information and Internet Policy*, October 2010.

[36] Youtube. http://www.youtube.com/my_speed as of November 2011.

[37] D. Croce, T. En-Najjary, G. Urvoy-Keller, and E. Biersack. Capacity estimation of adsl links. In *CoNEXT*, 2008.

[38] K. Cho, K. Fukuda, H. Esaki, and A. Kato. The impact and implications of the growth in residential user-to-user traffic. In *ACM SIGCOMM 2006*, 2006.

[39] M. Siekkinen, D. Collange, G. Urvoy-Keller, and E. Biersack. Performance limitations of adsl users: A case study. In *the Passive and Active Measurement Conference (PAM)*, 2007.

[40] Grenouille. Grenouille. http://www.grenouille.com/.

[41] C. R. Simpson Jr. and G. F. Riley. Neti@home: A distributed approach to collecting end-to-end network performance measurements. In *the Passive and Active Measurement Conference (PAM)*, 2004.

[42] Dongsu Han, Aditya Agarwala, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, and Srinivasan Seshan. Mark-and-sweep: Getting the inside scoop on neighborhood networks. In *Proc. Internet Measurement Conference*, Vouliagmeni, Greece, October 2008.

[43] Isposure. http://www.isposure.com as of November 2011.

[44] Neubot. http://www.neubot.org as of November 2011.

[45] Ne.me.sys. https://www.misurainternet.it/nemesys.php as of November 2011.

[46] H. Cui and E. Biersack. Trouble shooting interactive web sessions in a home environment. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, pages 25–30. ACM, 2011.

[47] Speedtest.net. http://www.speedtest.net as of November 2011.

[48] Glasnost: Bringing Transparency to the Internet. http://broadband.mpi-sws.mpg.de/transparency.

[49] SamKnows. Measure Your Broadband Accurately. http://testmyisp.com/.

[50] Tzyy Jane Lai and Chih Yung Chen. Virtual community and customer participations in user centric internet service ventures. In *Management of Engineering Technology, 2008. PICMET 2008. Portland International Conference on*, pages 1020 –1027, july 2008.

[51] M. Boari, A. Corradi, E. Lodolo, S. Monti, and S. Pasini. Coordination for the internet of services: A user-centric approach. In *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, pages 434 –441, jan. 2008.

[52] Skype. http://www.skype.com/ as of November 2011.

[53] S. A. Baset and H. G. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1 –11, april 2006.

[54] Saikat Guha, Neil Daswani, and Ravi Jain. An experimental study of the skype peer-to-peer voip system, 2006.

[55] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. *SIGCOMM Comput. Commun. Rev.*, 37:37–48, August 2007.

[56] E.P. Freire, A. Ziviani, and R.M. Salles. Detecting voip calls hidden in web traffic. *Network and Service Management, IEEE Transactions on*, 5(4):204 –214, december 2008.

[57] Google. http://www.google.com/talk as of November 2011.

[58] K. Suh, D. R. Figueiredo, J. Kurose, and D. Towsley. Characterizing and detecting skype-relayed traffic. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1 –12, april 2006.

[59] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. Skype video responsiveness to bandwidth variations. In *IN NOSSDAV*, 2008.

[60] D. Bonfiglio, M. Mellia, M. Meo, N. Ritacca, and D. Rossi. Tracking down skype traffic. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 261 –265, april 2008.

[61] Dario Rossi, Marco Mellia, and Michela Meo. A detailed measurement of skype network traffic, 2008.

[62] D. Rossi, M. Mellia, and M. Meo. Following skype signaling footsteps. In *Telecommunication Networking Workshop on QoS in Multiservice IP Networks, 2008. IT-NEWS 2008. 4th International*, pages 248 –253, feb. 2008.

[63] B. Sat and B.W. Wah. Analysis and evaluation of the skype and google-talk voip systems. In *Multimedia and Expo, 2006 IEEE International Conference on*, pages 2153 –2156, july 2006.

[64] Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. Quantifying skype user satisfaction. *SIGCOMM Comput. Commun. Rev.*, 36:399–410, August 2006.

[65] Youtube. http://www.youtube.com as of November 2011.

[66] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 15–28, New York, NY, USA, 2007. ACM.

[67] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 1–14, New York, NY, USA, 2007. ACM.

[68] M. Zink, K. Suh, Y. Gu, and J. Kurose. Characteristics of youtube network traffic at a campus network-measurements, models, and implications. *Computer Networks*, 53(4):501–514, 2009.

[69] Atif Nazir, Saqib Raza, and Chen-Nee Chuah. Unveiling facebook: a measurement study of social network based applications. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 43–56, New York, NY, USA, 2008. ACM.

[70] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM.

[71] Facebook. http://www.facebook.com as of November 2011.

[72] Flickr. http://www.flickr.com as of November 2011.

[73] Livejournal. http://www.livejournal.com as of November 2011.

[74] Orkut. http://www.orkut.com as of November 2011.

[75] S. Fernandes, R. Antonello, J. Moreira, C. Kamienski, and D. Sadok. Traffic analysis beyond this world: the case of second life. *Nossdav'07*, 2007.

[76] Chi-Anh La and Pietro Michiardi. Characterizing user mobility in second life. In *Proceedings of the first workshop on Online social networks*, WOSN '08, pages 79–84, New York, NY, USA, 2008. ACM.

[77] Matteo Varvello, Fabio Picconi, Christophe Diot, and Ernst Biersack. Is there life in second life? In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 1:1–1:12, New York, NY, USA, 2008. ACM.

[78] Tvants. http://www.tvants.com as of November 2011.

[79] Pplive. http://www.pplive.com as of November 2011.

[80] Sopcast. http://www.sopcast.com as of November 2011.

[81] Ppstream. http://www.ppstream.com as of November 2011.

[82] Thomas Silverston, Olivier Fourmaux, Alessio Botta, Alberto Dainotti, Antonio Pescapé, Giorgio Ventre, and Kavé Salamatian. Traffic analysis of peer-to-peer iptv communities. *Comput. Netw.*, 53:470–484, March 2009.

[83] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Y.-S.P. Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2102 – 2111 vol. 3, march 2005.

[84] Xiaojun Hei, Yong Liu, and K.W. Ross. Inferring network-wide quality in p2p live streaming systems. *Selected Areas in Communications, IEEE Journal on*, 25(9):1640 –1654, december 2007.

[85] K. Church, A. Greenberg, and J. Hamilton. On delivering embarrassingly distributed cloud services. *Hotnets VII*, 2008.

[86] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *In Proceedings of Cloud Computing and Its Applications [Online*, 2008.

[87] A. Dainotti, W. de Donato, and A. Pescapé. Tie: A community-oriented traffic classification platform. *Traffic Monitoring and Analysis*, pages 64–74, 2009.

[88] V. Jacobson, C. Leres, and S. McCanne. libpcap, lawrence berkeley laboratory, berkeley, ca. *Initial public release June*, 1994.

[89] S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, and M. Zekauskas. A one-way active measurement protocol (owamp). *draft-ietf-ippm-owdp-11. txt*, 2004.

[90] J. Prokkola, M. Hanski, M. Jurvansuu, and M. Immonen. Measuring wcdma and hsdpa delay characteristics with qosmet. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 492–498. IEEE, 2007.

[91] A. Botta, A. Dainotti, and A. Pescapè. Multi-protocol and multi-platform traffic generation and measurement. *INFOCOM 2007 DEMO Session*, 2007.

[92] V. Jacobson and S. Deering. Traceroute tool, 1989.

[93] V. Jacobson. Pathchar: A tool to infer characteristics of internet paths, 1997.

[94] P. Beyssac. Bing, a bandwidth measurement tool based on ping. http://www.cnam.fr/reseau/.

[95] A.B. Downey. Using pathchar to estimate internet link characteristics. In *ACM SIGCOMM Computer Communication Review*, volume 29, pages 241–250. ACM, 1999.

[96] B.A. Mah. pchar: A tool for measuring internet path characteristics. *http://www.kitchenlab. org/www/bmah/aSoftware/pchar/*, 1999.

[97] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, volume 134, 2001.

[98] R.L. Carter and M.E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance evaluation*, 27:297–318, 1996.

[99] S. Saroiu, P.K. Gummadi, and S.D. Gribble. Sprobe: A fast technique for measuring bottleneck bandwidth in uncooperative environments. In *IEEE INFOCOM*, page 1, 2002.

[100] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 905–914. IEEE, 2001.

[101] 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*. Citeseer, 2002.

[102] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and active measurement workshop*, volume 4, 2003.

[103] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *Selected Areas in Communications, IEEE Journal on*, 21(6):879–894, 2003.

[104] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 39–44. ACM, 2003.

[105] nuttcp. http://www.ppstream.com as of November 2011.

[106] Giacomo Bernardi and Mahesh K. Marina. Bsense: a system for enabling automated broadband census: short paper. In *Proc. of the 4th ACM Workshop on Networked Systems for Developing Regions (NSDR '10), June 2010.*, 2010.

[107] A. Botta, A. Dainotti and A. Pescapé. Multi-protocol and multi-platform traffic generation and measurement. IEEE INFOCOM, Demo session, May 2007.

[108] Qt libraries. http://qt.nokia.com/ as of November 2011.

[109] Cygwin. http://www.cygwin.com/ as of November 2011.

[110] Geoserver. http://geoserver.org/ as of November 2011.

[111] BitTorrent. http://www.bittorrent.com/.

[112] Measurement Lab. http://measurementlab.net.

[113] Linksys WRT54GL. http://www.linksysbycisco.com/IT/it/products/WRT54GL.

[114] NOX Box. http://noxrepo.org/manual/noxbox.html.

[115] Netgear wndr3700v2. http://www.netgear.com/home/products/wirelessrouters/high-performance/wndr3700.aspx as of November 2011.

[116] Cerowrt liux distribution. http://www.bufferbloat.net/projects/cerowrt as of November 2011.

[117] S. Seshan and H. Balakrishnan. netperf: Network Performance Utility. ftp://daedalus.cs.berkeley.edu/pub/netperf, 1996.

[118] Fping Program. ftp://networking.stanford.edu/pub/fping/, 1997.

[119] Shaperprobe. http://www.cc.gatech.edu/~partha/diffprobe/shaperprobe.html.

[120] K.L. Calvert, W.K. Edwards, N. Feamster, R.E. Grinter, Y. Deng, and X. Zhou. Instrumenting home networks. *ACM SIGCOMM Computer Communication Review*, 41(1):84–89, 2011.

[121] G. Aceto, A. Dainotti, W. de Donato, and A. Pescapé. Portload: taking the best of two worlds in traffic classification. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1–5. IEEE, 2010.

[122] Airodump-ng. http://www.aircrack-ng.org/doku.php?id=airodump-ng as of November 2011.

[123] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband internet performance: A view from the gateway. In *Proc. ACM SIGCOMM*, Toronto, Ontario, Canada, August 2011.

[124] Comcast FAQ. http://customer.comcast.com/Pages/FAQViewer.aspx?Guid=024f23d4-c316-4a58-89f6-f5f3f5dbdcf6, October 2007.

[125] R.M. Compton, C.L. Woundy and J.G. Leddy. Method and packet-level device for traffic regulation in a data network. U.S. Patent 7,289,447 B2, October 2007.

[126] Wondershaper. http://lartc.org/wondershaper/, 2002.