POLITECNICO DI TORINO

TÉLÉCOM PARISTECH

PhD Course in

• **Electronics and Communications Engineering**
ScuDo Doctoral School

• **Computer Science**
École Doctorale Informatique, Télécommunication et Électronique (EDITE) de Paris

Network Traffic Measurements

Applications to Internet Services and Security

Enrico Bocchi

Matricola: 210706
Identifiant National Étudiant: 0g5eup000f5

Thesis Supervisors
Prof. Marco Mellia – Politecnico di Torino
Prof. Dario Rossi – Télécom ParisTech

Contents

List of Figures

List of Tables

Abstract

The Internet has become along the years a pervasive network interconnecting billions of users and is now playing the role of collector for a multitude of tasks, ranging from professional activities to personal interactions. From a technical standpoint, novel architectures, e.g., cloud-based services and content delivery networks, innovative devices, e.g., smartphones and connected wearables, and security threats, e.g., DDoS attacks, are posing new challenges in understanding network dynamics.

In such complex scenario, network measurements play a central role to guide traffic management, improve network design, and evaluate application requirements. In addition, increasing importance is devoted to the quality of experience provided to final users, which requires thorough investigations on both the transport network and the design of Internet services.

In this thesis, we stress the importance of users' centrality by focusing on the traffic they exchange with the network. To do so, we design methodologies complementing passive and active measurements, as well as post-processing techniques belonging to the machine learning and statistics domains. Traffic exchanged by Internet users can be classified in three macrogroups: (i) Outbound, produced by users' devices and pushed to the network; (ii) unsolicited, part of malicious attacks threatening users' security; and (iii) inbound, directed to users' devices and retrieved from remote servers. For each of the above categories, we address specific research topics consisting in the benchmarking of personal cloud storage services, the automatic identification of Internet threats, and the assessment of quality of experience in the Web domain, respectively.

Results comprise several contributions in the scope of each research topic. In short, they shed light on (i) the interplay among design choices of cloud storage services, which severely impact the performance provided to end users; (ii) the feasibility of designing a general purpose classifier to detect malicious attacks, without chasing threat specificities; and (iii) the relevance of appropriate means to evaluate the perceived quality of Web pages delivery, strengthening the need of users' feedbacks for a factual assessment.

Résumé

L'Internet est devenu pendant les années un réseau répandu connectant milliards d'utilisateurs et joue maintenant le rôle de collecteur pour une multitude de tâches, en allant des activités professionnelles aux interactions personnelles. Du point de vue technique, des nouvelles architectures, par exemple des services basés sur le cloud et des réseaux de distribution des contenues, des appareils innovants, comme les smartphones et les dispositifs connectés, des menaces pour la sécurité, par exemple les attaques DDoS, posent de nouveaux défis pour la compréhension des dynamiques du réseau.

Dans ce scénario complexe, les mesures de réseau jouent un rôle central pour guider la gestion du trafic, l'amélioration de la conception du réseau et l'évaluation des demandes applicatives. De plus, une importance croissante est dévouée à la qualité d'expérience fournie aux utilisateurs finaux, qui demande des investigations soit sur le réseau de transport soit sur la conception des services Internet.

Dans cette thèse, nous soulignons l'importance de la centralité des utilisateurs en nous focalisant sur le trafic qu'ils échangent avec le réseau. A cet effet, nous concevons des méthodologies prévoyant des mesures passives et actives, ainsi que des techniques de post-traitement appartenant aux domaines de l'apprentissage automatique et des statistiques. Le trafic échangé par les utilisateurs Internet peut être classé en trois macro-groupes: (i) sortant, produit par les dispositifs des utilisateurs et envoyés au réseau; (ii) non sollicité, faisant partie d'attaques malveillantes menaçant la sécurité des utilisateurs; et (iii) entrant, dirigé vers les dispositifs des utilisateurs et téléchargé de serveurs distants. Pour chacune de ces catégories, nous abordons des sujets de recherche spécifiques constitués respectivement par l'analyse comparative des services de stockage sur cloud, par l'identification automatique des menaces Internet, et par l'évaluation de la qualité de l'expérience Web.

Les résultats comprennent plusieurs contributions dans le cadre de chaque sujet de recherche. En bref, ils éclaircirent (i) l'interaction parmi les choix de conception des services de stockage cloud, qui ont un impact important sur les performances fournies aux utilisateurs; (ii) la faisabilité de concevoir un

classificateur à usage général pour détecter les attaques malveillantes, sans se cibler sur les spécificités des menaces; et (iii) l'importance des méthodes appropriées pour évaluer la qualité perçue de la livraison des pages Web, en renforçant le besoin de feedback des utilisateurs pour une évaluation factuelle.

Chapter 1

Introduction

The Internet is undoubtedly a capillary and pervasive network where billions of people exchange information every day. In addition to interconnecting more and more users, it supports many and diverse applications related to business (e.g., e-commerce, Internet banking, finance and trading), professional activities (e.g., e-mail, videoconferencing, collaborative document editing), education (e.g., e-learning, crowd-sourced encyclopedia), entertainment (e.g., on-line gaming, video streaming), and personal interactions (e.g., instant messaging, social networking, photo sharing).

In its early days, the Internet was designed to fulfill the need of exchanging an exiguous amount of Bytes among a restricted set of research centers and governmental institutions. The supported services were few, working over rigidly-specified protocols, and mostly text-based (e.g., electronic mail, remote login, and file transfer). Even though the core building blocks (e.g., TCP/IP) did not change much since the 1960's Arpanet [1], the network now faces a plethora of services with different requirements [2, 3] and an ever-growing amount of traffic to be carried [4].

The variety of Internet applications is continuously expanding, considerably increasing the number of activities that users can conduct online. This has a severe impact on the network architecture and its internal complexity. In the last decade, we witnessed the development of novel network architectures, the wide-spreading of innovative devices, and the rise of new security challenges. Listing few examples: (i) Web applications [5] are now customary, with browsers being the de-facto means to access most of them; (ii) cloud infrastructures [6] are widely used to store and process huge amount of data; (iii) the rise of Content Delivery Networks (CDNs) [7] redefined the paradigm of content distribution and availability; (iv) mobile devices [8] are common and widely used to access Internet services; (v) cyber threats [9] are becoming

an urgent open issue and a source of troubles, given the amount of sensitive information being exchanged over the network.

As a consequence, researchers and professionals show a constantly renewed interest in understanding the dynamics interacting in such ever-evolving ecosystem. A precise and timely knowledge at the service of content providers and ISPs would allow a better network administration, an enhanced resilience against failures, a wiser usage of available resources, a timely reaction to new applications, and the provisioning of new services. Such aspects are of paramount importance today as the Internet is more and more driven by economic interests: Technical achievements bring improvements over elder technologies but their actual adoption depends on the usefulness to develop new business. Also, increasing attention is given to the Quality of Experience (QoE) offered to final users: Given the competitive reality in which they work, content providers and ISPs must always provide a satisfactory QoE.

Achieving a comprehensive understanding of network dynamics is not an easy task. At first sight, a deep knowledge of each application and of the traffic it produces seems highly desirable. However, this approach is not feasible for three main reasons: (i) The variety of applications and the constant rise of new services do not allow the development of per-application techniques; (ii) the majority of applications runs over encrypted connections or make use of proprietary protocols whose specifications are not publicly disclosed; (iii) the increasing usage of encrypted communications to protect users' privacy limits the visibility on application internals [10]. Therefore, there is the strong need to develop strategies that are not case-specific or dependent on the application under study. To achieve this, we aim at defining tools and methodologies able to infer meaningful statistics by leveraging the network traffic and different visibility aspects of an operational network. What we propose is thus to inspect network traffic at various levels, e.g., per-packet, per-flow, per-user, per-application, in order to gain a complete understanding of network activities and Internet services. The described process belongs to the field of Internet traffic monitoring and measurements [11, 12].

In this thesis, we focus on Internet users and we put the traffic that they exchange with the network at the core of our investigations. At the time of writing, industry reports [4] state that each Internet user owns more than two connected devices (which are expected to double in five years' time), ranging from PCs and laptops to smartphones, tablets, and smartwatches. These types of devices present different features (e.g., screen sizes, input/output interfaces, computing capabilities, energy resources, etc.) and connectivity characteristics (e.g., ADSL, WiFi, 3G/4G). Users are thus connected 24 hours a day, independently on the place where they are. More importantly, they

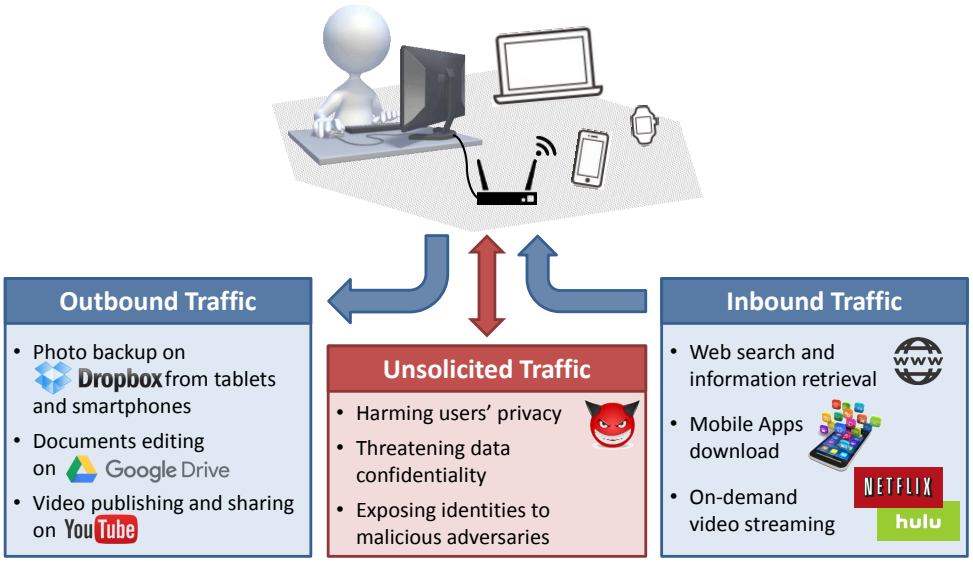


Figure 1.1: Categories of traffic exchanged by Internet users.

expect (and demand) a satisfactory experience even though they are likely unaware of the connection characteristics and of the application demands.

Fig. 1.1 summarizes the scenario in which we operate. All the devices belonging to a user are connected to a modem router, i.e., the home access gateway, through which they reach the Internet. The traffic that devices exchange can be summarized in three macro-categories:

1. **Outbound traffic** – It is the content being produced by users' devices that has to be pushed to the network, i.e., *upload traffic*. It includes photos shared on social networks or with friends, backup of videos produced with smartphones and tablets, documents being edited and shared among colleagues, etc.

2. **Unsolicited traffic** – It is the traffic produced by malicious adversaries targeting users' devices under various forms of attacks. It includes port-scans or net-scans, exploit kits hidden in Web pages, exploited vulnerabilities and security breaches, etc.

3. **Inbound traffic** – It consists of the content retrieved from the network and landing on users' devices, i.e., *download traffic*. It includes, for instance, objects needed to render Web pages, downloads of executables for PCs and Apps mobile devices, multimedia content provided by video and audio streaming services, emails fetched from mail servers, etc.

1.1 Topics and Research Questions

In this thesis, we aim at achieving visibility on all the three types of traffic listed above targeting specific research topics. Our goal is to take the user perspective and characterize the traffic produced by Internet services using both tailored approaches as well as generic algorithms applied to network traffic processing. We focus our attention on three main topics: (i) Benchmarking and comparison of Cloud Storage Services for personal usage, representative of *outbound* traffic; (ii) Detection and Characterization of Network Security Threats, representative of *unsolicited* traffic unwillingly received by users' devices; and (iii) Assessment of User Quality of Experience in the Web domain, representative of *inbound* traffic. In the following, we provide a brief description of each topic with the research questions we intend to address.

1.1.1 Benchmarking Personal Cloud Storage Services

In recent years, data storage became a fundamental service with companies, universities and private users having the need of storing large amounts of data. Cloud storage services are emerging as a strong alternative to local storage, allowing professional customers to save the costs and the burdens of buying and maintaining expensive hardware [13, 14], and attracting private enthusiasts to backup content with great simplicity and synchronize multiple devices seamlessly [15].

Despite the popularity of these services, very little is known about their features. Where is data stored? Which technologies are adopted to transfer users' file in a secure way? Which is the workload that the cloud infrastructure has to handle? Which traffic load has to be sustained by the network? Are these services efficient with ordinary usage operations?

Our contribution is the definition of a methodology designed to benchmark cloud storage services, unveiling their architecture, the provided performance, and the advanced capabilities implemented. The development of an active and configurable testbed allows the measurement of performance metrics using different workloads, ranging from synthetic ones, tailored to the identification of advanced capabilities, to realistic ones, useful for inferring service performance in real-life circumstances. In addition, we also provide a characterization of the typical usage of cloud storage services by means of passive traffic collection and analysis.

1.1.2 Network Security Threats

Information security over the Internet is becoming a key aspect due to the huge amount of personal information exchanged every day. Malicious adversaries are increasingly threatening users during ordinary Web surfing or mail exchange. Security researchers and practitioners in the field are taking different approaches to detect malware and provide countermeasures. For instance, they analyze the instruction set or the malicious code, study the behavior of an infected host in a controlled environment [16], i.e., a honeypot, or identify malicious connections to C&C botnets [17]. However, these approaches result in a set of methodologies focused on the specific malware being disassembled but are hard to generalize and make widely applicable. Cyber-attackers modify malware continuously using sophisticated schemes to evade detection by security tools. As result, existing antivirus software has a disappointing detection rate (<5%) of newly created viruses [18].

Some questions naturally emerge: Is there a way to model malicious activities in general, i.e., without being threat-specific? Is it possible to define a methodology that addresses the problem from a high-level perspective? How can we distinguish and characterize benign and malicious events showing a similar network behavior? Can we automate the process and spot new threats (e.g., zero-day attacks) automatically?

The idea is to design a methodology to process, extract and pinpoint network activities taking place with the occurrence of a malicious event. The methodology correlates such activities over time (i.e., analyzing different samples of traffic from the same host) and space (i.e., identifying common patterns among multiple hosts). Moreover, it aggregates information coming from different network layers (e.g., HTTP, DNS, TCP) to uncover hidden malicious behaviors. The result is an enhanced visibility on the incident that can be framed into an easily understandable graphical representation. In turn, the output of the process can be modeled through the definition of features enabling the training of a classifier aimed at distinguishing between benign and malicious events. Potential applications are numerous, from helping in forensic analysis to the ability to spot new malicious attacks showing patterns similar to the known ones.

1.1.3 Web Quality of Experience

HTTP is the most popular application layer protocol of today's Internet, being the de-facto solution for the deployment of new services and applications. However, it is used to accomplish tasks for which it was not designed, resulting in inefficiencies and performance limitations [19]. Only recently,

researches and practitioners proposed improvements like SPDY [20] (now standardized by the IETF as HTTP/2 [21]), a novel protocol designed to reduce the latency for accessing Web content, and QUIC [22], an improved version of HTTP using UDP at Transport Layer (L4).

Despite the high interest, a deep analysis evaluating benefits and drawbacks of the newcomers from the user perspective is still missing. Is HTTP/2 reducing the time needed to load the page and improving users' QoE? Are users able to distinguish between the two HTTP revisions, ultimately electing the best performing protocol from the experience standpoint? To what extent the performance provided by HTTP/2 are page dependent? Is it possible to estimate users' QoE from performance indicators like the page loading time or the time needed to render content on screen?

Given the novelty and the potential wide adoption of the new protocols, several aspects are worth to be investigated. Our contribution consists of a detailed survey of the commonly used indicators to evaluate Web performance and the collection, by experimental means and laboratory sessions, of actual users' feedbacks, i.e., Mean Opinion Score (MOS). By contrasting MOS points and performance indicators, we aim at understanding which protocol between HTTP/1 and HTTP/2 is the best performing and to what extent actual user feedback and objective indicators are correlated.

1.2 Approach

The research work carried out in this thesis has its foundations in Internet traffic measurements [23]. Nowadays communication networks play the role of collectors for the traffic produced by users against Web pages, Internet services, and cloud-based applications. Therefore, having the ability to capture and characterize the traffic opens for Internet services popularity understanding, users' habits unveiling, and performance assessment.

Fig. 1.2 shows the workflow used to achieve visibility on research subjects starting from network measurements. Such workflow is made of three steps: (i) *Measurements*, in charge of extracting packets from the raw network traffic and store them in a repository; (ii) *Filtering and Aggregation*, consisting of a set of techniques aimed at post-processing the collected traffic; and (iii) *Analytics*, responsible for the understanding and the interpretation of produced data.

In more details: *Raw Data Acquisition* corresponds to the act of capturing network traffic with a per-packet granularity. We intend to perform traffic captures complementing [24] *active measurements*, i.e., proactive probing of Internet services, and *passive measurements*, i.e., recording traffic without

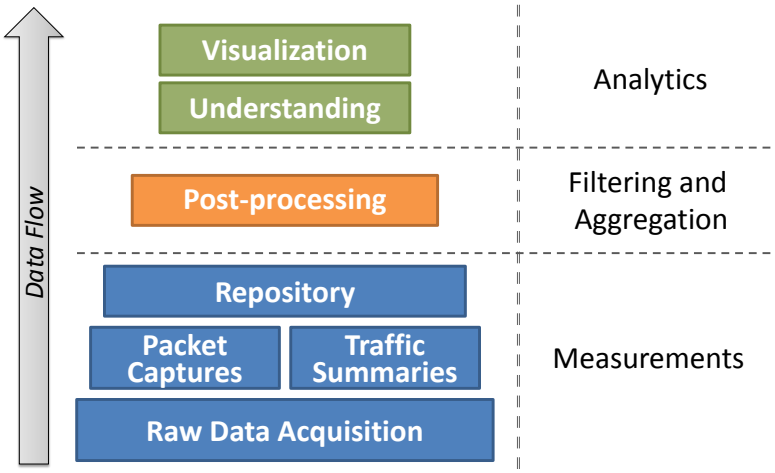


Figure 1.2: Network measurement and data processing workflow.

interfering with it. Collected packets can be stored for archival or processing purposes in the form of *Packet Captures*, i.e., exactly as they were captured, or be condensed in *Traffic Summaries*, i.e., combining the information coming from multiple packets in a single record. In both cases, the produced output is moved to a central *Repository* for storage.

On top of the storage system, network data is processed by the *Filtering and Aggregation* step, which allows one to access the recorded traffic samples in a simple and scalable way and to quickly output the desired data portions. This is possible thanks to innovative technologies like Apache Hive [25], which facilitate the access to large datasets over distributed storage.

Finally, the *Analytics* step employs techniques and methodologies coming from a variety of domains, including reverse engineering, data mining, machine learning, statistics, etc. Here the desired data is interpreted for extracting knowledge and reporting purposes. The potential applications are numerous, ranging from the identification of bottlenecks or impairments in connectivity to the assessment of provided performance. Being the understanding of collected data samples a problem-specific task, it is of paramount importance to use the most suitable techniques according to the context, the available dataset, and the question being targeted.

We now report a description of the tools at different steps of the processing workflow (see Fig. 1.2). They consist of either specialized techniques for network traffic processing or in general purpose algorithms applied to specific contexts. In the latter case, additional details are reported along with the dissertation of the research topic for which they are applied.

1.2.1 Passive Traffic Measurements

Passive traffic collection refers to a set of activities aimed at recording network traffic as it passes through links and devices without interfering with it. Capturing traffic requires the setup of a probe instrumented to sniff raw packets while they are flowing through the network. Probes can be either specialized hardware deployed on purpose or be built into already-existing network devices [26] (e.g., routers and switches).

Passive measurements have the advantage of being potentially performed on real and operational networks installing probes at the border router of a campus network, at the Point of Presence (PoP) of an Internet Service Provider (ISP) and on links gathering at an Internet Exchange Point (IXP). This results in collecting traffic that is highly representative of actual users' activities, that can be helpful in estimating Internet services diffusion and popularity, as well as for network troubleshooting purposes [27, 28].

However, passive measurements present some limitations. For instance, the collected traffic is bound to the time and the place where the capture is performed. Thus, the emerging picture might be biased or limitedly representative for the population under study. In addition, when used for troubleshooting, passive measurements might not provide the means to identify the root cause of the malfunction or the location/device producing the issue.

In this thesis work, passive measurements are employed to characterize the usage of Cloud Storage Services (Sec. 3), to spot malicious activities produced by Internet threats (Part II), and to assess the impact of Carrier Grade NAT on users' traffic (Sec. 11). To record network packets we rely on ad-hoc deployed probes running Tstat and, additionally for malware analysis, we process the traffic using two independent Intrusion Detection Systems. Both tools are detailedly described in what follows.

Network Traffic Analyzer – Tstat

TCP STatistic and Analysis Tool (Tstat) [29] is an advanced modular open-source tool for passive network traffic capture and analysis whose development has to be attributed to the Telecommunication Network Group of the Politecnico di Torino, Italy. Through several updates and improvements implemented along the years, Tstat is now a sophisticated piece of software able to monitor live networks up to 40 Gb/s speed on commodity hardware [30]. It provides traffic classification capabilities through behavioral classifiers [31], high-level visibility on encrypted traffic through the analysis of Domain Name System (DNS) queries [32], and a through characterization of on-going activities in the monitored network.

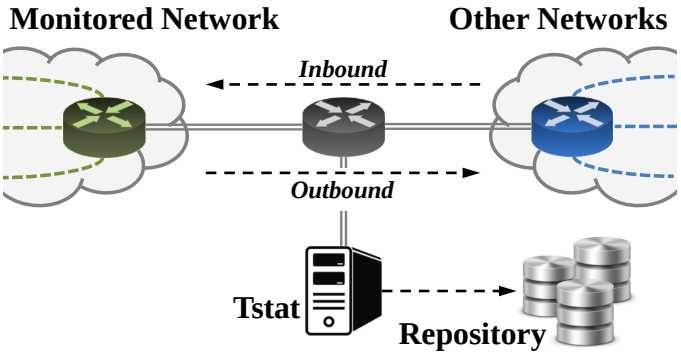


Figure 1.3: Tstat deployment scenario.

Fig. 1.3 shows the typical deployment scenario for Tstat. On the left-hand side, a network to be monitored, e.g., a campus network, is connected to a border gateway through which it reaches other networks (right-hand side) including, for instance, the public Internet. The two networks are connected via an access link where both incoming and outgoing packets are flowing. A machine is instrumented to sniff the traffic on the link, thus playing the role of a passive probe, and is equipped with Tstat, which in turn processes the traffic to extract the desired pieces of information.

The basic operation performed by Tstat consists of processing the Internet Protocol (IP) packets flowing on the monitored link to rebuild upper-layer flows. Such task is performed by grouping packets according to precise rules that define a flow identifier, $flow_{ID}$. A typical choice is to group packets according to the tuple defined by ($Proto$, IP_{src}, $Port_{src}$, IP_{dst}, $Port_{dst}$), where $Proto$ is the protocol used at the L4, i.e., Transfer Control Protocol (TCP) or User Datagram Protocol (UDP). In addition, as IP addresses and ports can be reused over time, a $flow_{ID}$ uniquely identifies a traffic flow for a limited period of time. Being TCP a connection oriented protocol, the beginning and the end of a TCP flow are provided by the identification of the connection set-up and tear-down messages, i.e., `SYN` and `FIN` flags set in the TCP header, respectively. In case the connection is abruptly interrupted without the transmission of `FIN` messages, the flow is considered closed after an idle time during which no packets are transmitted. For UDP, instead, a flow is identified when the first packet matching a new $flow_{ID}$ is detected in the network and considered closed after an idle time.

By rebuilding TCP and UDP flows, Tstat is able to provide a rich set of statistics, some of which are common to all flows, e.g., source and destination IP address, number of bytes and packets exchanged, timestamp of

the first packet seen, and connection duration. Other statistics are instead conditioned to the protocol used at the L4. While for UDP only the source and destination port numbers are reported, TCP statistics provide counters for TCP flags, i.e., `SYN, ACK, FIN, RST`, number of retransmitted bytes and packets, and timestamps for first and last packet with payload, etc.

Through additional modules, it is possible to have Tstat extracting an extended set of statistics for specific Internet services including Peer-To-Peer (P2P) (e.g., eMule), video streaming (e.g., generic streaming and YouTube), instant messaging (e.g., Yahoo! messenger, MSN messenger), voice calls and video conferencing (e.g., Skype). Tstat is also able to parse application-specific information from HyperText Transfer Protocol (HTTP) and DNS protocol headers, which are of paramount importance for this work.

In the context of HTTP flows, Tstat extracts directly from HTTP headers [33] the number of HTTP requests and responses, the full Uniform Resource Locator (URL) of the requested object, the content-type, the content-length, the user-agent of the browser (or of the application generating the request), the response status, etc.

Tstat also embeds pieces of information coming from the DNS protocol that are precious to have a hint on the Internet services producing encrypted traffic. Indeed, with the development and integration of DN-Hunter [32], Tstat snoops DNS responses provided by DNS resolvers, annotating TCP and UDP flows with the server Fully Qualified Domain Name (FQDN). Additionally, it reports the IP address of the DNS resolver, the response code [34], and estimates the time needed to resolve the hostname.

Overall, Tstat produces more than 100 (19) statistics for TCP (UDP) flows.[1] Among the various output formats offered, flow-level logs are primarily used for this work. They consist of textual files arranged in tables, where each traffic flow is stored in a record, i.e., a row of the table, and each column reports a specific statistic of the flow, source and destination IP, amount of bytes exchanged, connection duration, etc. Multiple files are generated by Tstat according to the configuration used. Generally speaking, one file for TCP traffic, a second for TCP connections for which the three-way handshake is not completed, and a third for UDP traffic are produced. Additional files reporting about, e.g., video streaming or Skype services might be produced. Log files generated by Tstat are then collected and stored in a central repository (lower-right part of Fig. 1.3) for off-line post-precessing.

[1]The exact amount of statics reported for TCP flows depends on the actual Tstat configuration and modules executed at run-time. A complete list is available at `http://tstat.tlc.polito.it/measure.shtml#LOG`

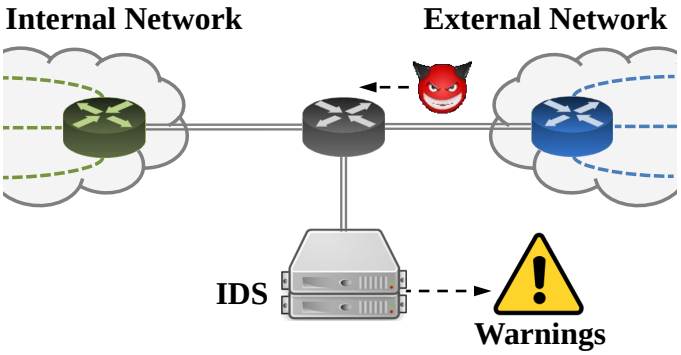


Figure 1.4: IDS deployment scenario.

Intrusion Detection Systems

An Intrusion Detection System (IDS) is a hardware device or a software that monitors a system for detecting suspicious activity and records the evidence of an intrusion. Monitored activities are considered legitimate or suspicious according to a set of internal rules which can either define specific patterns (i.e., signature-based detection) or be based on the identification of generic anomalous conditions (i.e., anomaly-based detection) [35].

Depending on the object being monitored and on the deployment location, IDSes can be classified as Network IDS (N-IDS) or Host IDS (H-IDS). N-IDSes analyze network traffic looking for anomalous or illicit behaviors and flag suspicious activities matching any internal detection rules. Their monitoring activity applies to the whole network they are connected to and are thus able to spot suspicious behaviors threatening all the devices connected to the same network. On the other hand, H-IDSes tackle the same problem but their monitoring activity is limited to the scope of a single host. In this work, we focus solely on N-IDS.

Independently on the type and on the detection technique, the role of an IDS is limited to identification and reporting, i.e., IDSes do no block anomalous activities autonomously. To achieve protection it is thus needed to forward warnings raised by IDSes to firewalls or intrusion prevention systems, which in turn block the matching network traffic.

Fig. 1.4 shows the typical operation scenario of a N-IDS. Similarly to Tstat, the IDS is capable of analyzing network traffic that is mirrored through a hub or a span port. The output produced is organized on a per-event basis and reports a flow identifier, $flow_{ID}$, made of ($Proto$, IP_{src}, $Port_{src}$, IP_{dst}, $Port_{dst}$) together with additional information like the timestamp at which the event was recorded, the application layer protocol, a priority grade, etc.

Besides the $flow_{ID}$, the most important field reported is the $threat_{ID}$, i.e., a numerical code that indicates the detection rule matched by the event. In most cases, IDSes provide along with the numerical identifier a description of the threat. Such information can be valuable to the security analyst or for deciding whether to enforce block/allow rules through a firewall.

In this work we use two IDSes: (i) A Symantec non-commercial product for big-sized networks; and (ii) Snort [36]. In our use case, the two IDSes run in parallel and analyze the same network traffic. In addition, they both provide information about the threats detected through $threat_{IDs}$ and related description. However, while in the case of Snort $threat_{IDs}$ are entirely documented[2], the threat description provided by Symantec is limited to some families of threats only and the detection rules are never reported. For our purposes, we consider IDSes as *oracles* able to identify malicious activities in the network traffic. That is, events in the traffic for which an IDS raises an alarm are considered as malicious.

1.2.2 Active Traffic Measurements

Active traffic collection refers to the activity of injecting packets into a network and consequently study how the traffic reaches its destination and which are the produced consequences, i.e., identify the cause – effect relationship between injected packets and observed reactions. Packets can target the network itself to identify eventual connectivity issues or be directed to servers and Internet applications to measure the offered service.

Packets injected in the network are specifically crafted for the problem being addressed. This implies flexibility and explicit control of the whole testing process, as well as the possibility to calibrate tests on the need. That is, performing active measurements enables researchers and practitioners to run the required tests when these are needed. In addition, active measurement techniques collect a smaller amount of traffic compared to the humongous data volume recorded by passive means. This results in a straight-forward understanding of the gathered traffic and in an easier post-processing.

Active measurements, however, come at a cost. The very same process of generating and injecting packets implies an increase of the network load that, in extreme cases, might cause burdening of resources and negatively impact on users' traffic. In addition, packets being produced depend on parameter settings that hardly replicate real conditions or, for doing so, would require an additional effort to understand and model a real-life scenario.

[2]Snort rules can be found at `https://www.snort.org/downloads/#rule-downloads`

More importantly, the employment of active measurements requires the design and the implementation of testbeds through which experiments can be performed. This stems from the specificity of the problem being tackled that, in turn, requires the execution of controlled and customizable experiments. Deploying dedicated testbeds constitute a non-negligible engineering effort that is however paid back by the acquired flexibility and by the complete control of experiment execution.

Active measurements have been the subject of various projects carried out by the research community [37,38,39,40] ranging from the assessment of link properties (e.g., latency, loss probability, available bandwidth) and the Quality of Service provided by DNS resolvers, to the discovery of the Internet network topology. Also the industrial world implemented products [41,42,43] targeting performance evaluation and connectivity issue detection.

In this work, active measurements are used to unveil architectures, capabilities, and performance of Cloud Storage Services (Sec. 4). Similarly, active probing is employed for characterizing metrics used in Web Quality of Experience evaluation (Sec. 10.2) and for collecting Mean Opinion Score grades in controlled conditions (Sec. 10.3). In both cases, specialized testbeds are designed for the purpose. A thorough description of testbeds design and implementation is provided in the respective sections.

1.2.3 Analytics – Deep Packet Inspection

Deep Packet Inspection (DPI) is a network traffic analysis technique consisting in the examination of the Application Layer (L7) payload, widely used for both open-source [44] and commercial [45] products. The use of DPI allows the identification of specific pieces of information or patterns in the payload that are helpful for traffic filtering and classification purposes. DPI can be called to accomplish more complex tasks if the payload analysis is performed on sequences of packets, instead of being carried out on each single packet in isolation. This enables the detection of more complex patterns and transactions that can be modeled through finite-state machines, e.g., request and response mechanism in HTTP transactions.

DPI techniques, however, are not applicable when traffic encryption or obfuscation is used. In case of HTTP Secure (HTTPS) transactions, for instance, Secure Sockets Layer (SSL)/Transport Layer Security (TLS) is used on top of TCP causing the payload to become a sequence of random numbers and thus inhibiting the recognition of patterns in the payload. This is indeed a limitation of DPI techniques that might impede their utilization in a variety of fields, given the ever increasing usage of encryption [10].

In this thesis, DPI is used for post-processing the traffic traces generated by Cloud Storage Services (Sec. 4). As we shall see, the software client supplied by cloud providers cannot be instrumented to extract performance metrics. For this reason, the approach followed consists in recording the traffic produced by cloud storage services and in extracting statistics (e.g., volume of data exchanged, timestamps, etc.) from it at a later stage. Despite most of the traffic is encrypted, DPI is still able to provide a satisfactory degree of granularity. Indeed, thanks to string matching on SSL/TLS certificates, it is possible to classify traffic flows and infer whether they carry users' payload (i.e., storage flows), or control traffic (i.e., control flows). It is also possible to detect sequences of events working as file or chunk delimiters.

The employment of DPI techniques, however, requires a non-negligible effort in understanding the dynamics of the network traffic. More specifically, ingenuity is required in *reverse engineering* the typical behaviors and the fingerprints of the applications under study. While this might not be an issue with well-established and standardized protocols, applications making use of proprietary or customized protocols might easily break the defined rules leveraged by DPI, compromising patterns identification and ultimately making the whole process ineffective. For such reason, workflows based on DPI require continuous maintenance and revision efforts, which hardly scale due to the variety of Internet services and their frequent updates.

1.2.4 Analytics – Machine Learning

In this thesis, some of the post-processing techniques used come from the machine learning community [46]. We focus on supervised learning techniques and, more specifically, on data classification, i.e., the process of allocating data instances to specific categories. In this class of techniques fall Naïve Bayes classifiers, decision trees algorithms (e.g., C4.5, Random Forest, etc.), and Support Vector Machines [47]. Supervised data classification demands the availability of a *training set*, i.e., a pre-labeled dataset with which the classification algorithm can be trained. The training set has to be representative of the desired output. That is, it has to contain all the output classes in which new data samples have to be classified. Building a comprehensive training set is not a trivial task which mandates manual inspection and labeling or the election of an oracle to build an appropriate ground truth.
The process allowing data classification can be summarized in four steps.

1. Definition of the classification object.
The classification object is the data instance that has to be put in one of the output categories by the classifier. The nature of objects can greatly vary

according to the application scope. In this work, objects being classified are Connectivity Graph (Sec. 8).

2. Feature vector extraction.
Classification algorithms require data instances to be represented by a feature vector. The process of extracting the feature vector consists in summarizing data instances through a set of measurable properties, each called *feature*. Features can be binary values (e.g., True or False), numerical values, nominal values (i.e., the list of values a feature can assume is known), or textual strings. Features to be considered for a successful classification are problem dependent and usually not known a-priori.

The definition of an adequate feature vector is of primary importance for the quality of the classification process. Features need to be as informative as possible so to help the algorithm in properly classifying data instances. A second aspect to be considered is the dimension of the feature vector. Leveraging tens or, worse, hundreds of features can be counterproductive due to the hidden correlations existing among them, which can potentially confuse the learning algorithm making it inefficient.

To solve both issues, specific techniques have been defined to identify the most relevant features out of all the available ones in the feature vector. The process of selecting useful features is called *feature selection* and aims at simplifying the classification model, in turn mitigating potential overfitting issues, as well as at reducing the training time by limiting the dimensionality of the input space. Among all feature selection techniques, we use Minimum Redundancy, Maximum Relevance (mRMR) [48], considered state-of-the-art.

3. Classification process.
The decision process consists in defining the output category of each data instance being processed by the classifier and can be regulated by a variety of algorithms. In this work, we make use of tree-based classifiers, i.e., decision trees and random forest classifiers. Both types of classifiers create a decision model based on interior nodes, which represent the selected classification features, and leaves, which represent the outcome of the decision process. Branches connect each parent node to children nodes according to the values that can be assumed by the feature embodied by the parent node. Branches are thus the links connecting the head of the tree to the outcome or, in other words, the sequence of features that lead to the classification decision.

The tree-based classification has several advantages when compared to other algorithms. For instance, it produces a model that can be easily visualized and interpreted, enabling the verification by domain experts, supports multi-category classification, and processes large datasets on commod-

Table 1.1: Example of confusion matrix.

		Predicted	
		YES	NO
Actual	YES	True Positive	False Negative
	NO	False Positive	True Negative

ity hardware in a reasonable time. On the other hand, decision trees are subject to overfitting problems, i.e., creating a classification model which is not general enough and rather tailored to the data being used as training set. Such issue can be mitigated limiting the maximum depth of a tree during its construction process, or pruning the tree once it has been defined by removing parts that marginally contribute to the classification accuracy.

4. Performance assessment.
Once a classification model has been produced, its performance has to be evaluated. To do so, a *testing set* is required. Testing sets are collections of data instances whose categories are already known. They play the role of ground truth and can be obtained either by using *oracles* able to provide category labels or by creating artificial ones.

To assess classification accuracy, the model defined through the training set is applied to the testing set and the achieved results are contrasted against the ground truth. Limiting the discussion to *binary classifiers*, which classify data instances as either belonging to the specific class (i.e., positives) or not (i.e., negatives), four outcomes are possible:

1. True Positive (TP) (i.e., hit) – the classifier and the ground truth are in agreement and the data instance belongs to the class;

2. False Positive (FP) (i.e., false alarm) – the classifier considers the data instance as belonging to the class but the ground truth does not;

3. True Negative (TN) (i.e., correct rejection) – the classifier and the ground truth are in agreement and the data instance does not belong to the class;

4. False Negative (FN) (i.e., miss) – the classifier considers the data instance as not belonging to the class but the ground truth does.

The four possible outcomes are usually reported in the form of *confusion matrix*, which represents the data instances in the predicted category by the classification model on columns and the data instances in the actual category

(i.e., the category known from the ground truth) on rows. An example of confusion matrix is shown in Tab. 1.1.

Possible outcomes can also be combined in more advanced metrics:

- **Precision**, also known as Positive Predicted Value (PPV) – It is a measure of the data instances actually belonging to the class out of all the returned results. It indicates the fidelity of the classification.

$$PPV = \frac{TP}{TP + FP}$$

- **Recall**, also known as Sensitivity or True Positive Rate (TPR) – It is a measure of the data instances belonging to the class that are correctly classified as such. It indicates the completeness of the classification.

$$TPR = \frac{TP}{TP + FN}$$

- **F-Measure** – It combines Precision and Recall in a single metric. The most commonly used version of the F-Measure, called balanced F-score or F_1 score, attributes equal weight to Precision and Recall resulting in the harmonic mean of the two.

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- **Accuracy** – It evaluates the number of correctly classified data instances, both TP and TN, out of all the available ones. It is a measure of the ability of the classifier to identify or exclude a condition.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

1.2.5 Application Domains

Tab. 1.2 summarizes the research topics addressed in this work, detailing the measurement methods used, the techniques employed to post-process collected traffic, and the potential applications of the analysis carried out.

Overall, both active and passive measurements are used in a complementary way for all topics with the only exception of Network Security. In this latter case, indeed, we prefer to rely exclusively on passive measurements in order to capture the variety of threat families available on the Internet today, which is hardly replicable in a controlled environment.

Table 1.2: Research topics and adopted methodologies.

Topic	Measurements	Analytics	Applications
Cloud Storage	Active Passive	Deep Packet Inspection	Performance Evaluation Design Guidelines
Network Security	Passive	Data Mining Data Classification	Augmented Visualization Malware Classifier
Quality of Experience	Active Passive	Statistics	Performance Evaluation Anomaly Detection

For what concerns post-processing methodologies and potential applications of the research, these greatly vary according to the topic. While understanding and visualization of the collected information are common to all, Cloud Storage Benchmarking and Quality of Experience assessment are more related to performance evaluation. The effort done in the scope of Network Security is instead directed to the definition of techniques and methodologies with which it is possible to identify malicious traffic among the humongous flow of legitimate activities that crowd the network.

1.3 Thesis Organization

Besides this introduction, the thesis is organized in three parts.

Part I – Personal Cloud Storage will focus on the usage characterization of personal cloud storage services and on the benchmarking of 11 offers available on the market. It is divided in three chapters: • *Chapter 2 – Cloud Storage Services* introduces the topic, mentions the related work available in the literature, and states the provided contributions; • *Chapter 3 – Personal Cloud Storage – Usage Patterns Characterization* characterizes the usage of cloud storage leveraging passive measurements and provides insights on access frequency to the service, data volumes exchanged, eventual dependencies with the type of terminal used (e.g., client running on PCs, mobile phones, Web interface); • *Chapter 4 – Personal Cloud Storage – Benchmarks and Comparison* complements the above results by means of active experimentations aimed at contrasting the performance offered and the advanced capabilities eventually implemented by 11 storage offers.

Most of this work has its roots in the following papers:
• E. Bocchi, I. Drago, and M. Mellia, "Personal Cloud Storage: Usage, Performance and Impact of Terminals," in *IEEE 4th International Conference*

on Cloud Networking, IEEE CloudNet 2015, pp. 106–111, Oct 2015.
• E. Bocchi, I. Drago, and M. Mellia, "Personal Cloud Storage Benchmarks and Comparison," in *IEEE Transactions on Cloud Computing*, IEEE TCC, Accepted for publication, 2015.

Part II – Network Security Threats will focus on the problem of malware over the Internet and, more specifically, on how to detect traffic generated by malicious adversaries. It is divided in four chapters: • *Chapter 5 – Network Security Threats* introduces the problem of information security and describes the current techniques to identify malware. It also highlights limitations and drawbacks of such techniques, introducing our novel approach and listing contributions; • *Chapter 6 – A View of Malware in Residential Networks* depicts the characteristics of malware identified in home networks by means of passive traffic collection and analysis through a reference IDS; • *Chapter 7 – MAGMA – MultilAyer Graphs for MAlware detection* introduces MAGMA, an automatic systems aimed at identifying network events that are highly correlated with activities known to be malicious. MAGMA outputs the collected results in the form of Network Connectivity Graphs (CGs), which constitute a compact representation of malicious traffic of great value to the security analyst; • *Chapter 8 – MAGMA – Supervised Classifier* builds on top of CGs introducing the use of machine learning techniques to distinguish between malicious CGs and benign ones. The result is the MAGMA classifier, which proves to have very high accuracy and to be resilient to parameter setting.

Most of this work has its roots in the following papers:
• A. Finamore, S. Saha, G. Modelo-Howard, S.-J. Lee, E. Bocchi, L. Grimaudo, M. Mellia, and E. Baralis, "Macroscopic View of Malware in Home Networks," in *IEEE 12th Annual Consumer Communications and Networking Conference*, IEEE CCNC 2015, pp. 262–266, Jan 2015.
• E. Bocchi, L. Grimaudo, M. Mellia, E. Baralis, S. Saha, S. Miskovic, G. Modelo-Howard, and S. J. Lee, "Network Connectivity Graph for Malicious Traffic Dissection," in *24th International Conference on Computer Communication and Networks*, ICCCN 2015, pp. 1–9, Aug 2015.
• E. Bocchi, L. Grimaudo, M. Mellia, E. Baralis, S. Saha, S. Miskovic, G. Modelo-Howard, and S.-J. Lee, "MAGMA: Network Behavior Classifier for Malware Traffic," in *Elsevier Computer Networks – Special Issue on Traffic and Performance in the Big Data Era*, vol. 109, Part 2, pp. 142–156, 2016.

Part III – User Quality of Experience will focus on the assessment of QoE for Web users considering objective performance metrics that can be computed instrumenting the browser or measuring network timings, as well

as directed feedback provided by real users, i.e., MOS grades. It is divided in three chapters: • *Chapter 9 – Web Quality of Experience* introduces the problem of quantifying the Web Quality of Experience (WebQoE) by automatic means. It also highlights the limitations of today's common practices in estimating WebQoE, and outlines our contributions; • *Chapter 10 – Assessing WebQoE* provides a comprehensive characterization of metrics used for WebQoE evaluation and the results of extensive experimentations carried out over the top-100 Alexa Web pages. In addition, through the development of a dedicated testbed, voluntaries are asked to watch pages being loaded under controlled conditions and rate their experience providing a MOS grade; • *Chapter 11 – Impact of Carrier Grade NAT* completes the picture by verifying the impact of Carrier Grade Network Address Translation (NAT) devices (middle boxes deployed in the network by ISPs on which final users have no ability to intervene) on Web traffic.

Most of this work has its roots in the following papers:
• E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the Quality of Experience of Web Users," in *Proceedings of the 2016 Workshop on QoE-based Analysis and Management of Data Communication Networks*, ACM SIGCOMM Workshops – Internet-QoE '16, pp. 37–42, Aug 2016.
• E. Bocchi, A. S. Khatouni, S. Traverso, A. Finamore, M. Munafò, M. Mellia, and D. Rossi, "Statistical Network Monitoring: Methodology and Application to Carrier-Grade NAT," in *Elsevier Computer Networks – Special Issue on Machine learning, data mining and Big Data frameworks for network monitoring and troubleshooting*, vol. 107, Part 1, pp. 20–35, 2016.

Finally, **Chapter 12 – Conclusions** summarizes this work, recaps the collected findings, and highlights the most significant results obtained.

In addition, **Part IV – Appendixes** reports additional details that are kept out of the main flow of this work to improve readability.

Part I

Personal Cloud Storage

Chapter 2

Cloud Storage Services

2.1 Introduction

Personal cloud services have become very popular among Internet users. The large amount of space they offer, the opportunity of synchronizing devices seamlessly, and the possibility of sharing files with other users keep attracting customers to the cloud. They are also a popular means for collaborative work, with files being automatically distributed and synchronized among colleagues, thus posing additional near real-time constraints.

More and more people are attracted by these services, saving personal files, synchronizing devices and sharing content with great simplicity. The high public success has pushed dozens of providers to the cloud storage market. Well-established players like Dropbox are now facing the competition with giants like Apple, Microsoft, and Google offering well-integrated solutions into Mac, Windows or Android Operating System (OS) and providing a large amount of storage space for cheap prices [13].

While the high competition for customers continues to lower the cost per GB [49], other important aspects, such as synchronization performance, are mostly unknown given the proprietary design of most services. As such, selecting the service that suits the specific needs of a user has become a non-trivial task. In the ever increasing competition for customers in the crowded market, it has to be expected that performance will be as important as price to attract end users in a near future.

Both new providers competing against established players and users migrating files to the cloud would greatly benefit from a deeper knowledge of personal cloud storage. Providers would take advantage of knowing what usage is made of cloud storage by end users, while those same users would be able to take an informed choice on the service better suiting their needs.

In addition, personal cloud storage services are data-intensive applications exchanging their payload on the public Internet. It is therefore required to carefully design the synchronization protocols in order to reduce bandwidth wastage and maximize the efficiency of the transfer. Badly designed services would inject additional load in the network, producing more traffic, and likely performing poorly from the user perspective.

A systematic methodology and comprehensive benchmarks that could assist in evaluating cloud storage offers are still lacking. In particular, little work has been done in understanding thoroughly users' behaviors and habits, the effects of design choices, and the implications of both on final performance. Shedding light on such aspects would enable engineers and practitioners in the field to improve service performance, identify and address bottlenecks, and enhance end users' satisfaction.

2.2 Related Work

Given the pervasiveness of personal cloud storage, it is not surprising that the research community puts a relevant effort in understanding such services. Several related works are based on active measurements. Authors of [50] were the first to compare storage providers. By manually performing a simple set of active experiments, they measure the backup time of given workloads for four different services, namely Dropbox, Mozy, Carbonite, and CrashPlan.

Other works [15,51,52], instead, focus on the characterization of a specific cloud storage service from passive measurements obtained in the network. These measurements are used to build an initial model for the Dropbox client behavior in [53]. Security threats for cloud storage services are discussed in [54, 55], in which the internal functioning of Dropbox is also revealed by means of reverse engineering in a lab environment.

The authors of [56] present a study similar to the one carried out in this work, but it is limitedly focused on server infrastructure. In [57], a performance analysis of the Amazon Web Services is presented, but it is not focused on personal storage. In [58,59], a set of active experiments is performed with Dropbox, and high traffic overhead is identified in certain scenarios. Authors of [60] propose a methodology to benchmark cloud storage and evaluate three services in a long-term measurement campaign. The methodology in [60], however, relies on public Application Programming Interfaces (APIs) to perform the measurements, thus it does not consider the eventual benefits or limitations of the synchronization client running on users' machines. Finally, the type of content people store in the cloud [61], and pricing practices in the storage market are discussed in [13].

Users' satisfaction in cloud storage has been studied in [62,63]. Technical aspects such as network bandwidth and synchronization latency are identified as important sources of users' satisfaction. In [64], a mechanism to ensure consistency between the local file system and the remote repository is proposed, possible improving the perceived quality of service. The characteristics of files stored in cloud services and possible bottlenecks in storage protocols have been analyzed in [15, 61, 65].

2.3 Contributions and Work Organization

In contrast to the works cited above, this part of the thesis presents two complementary efforts towards an all-round understanding of personal cloud storage services. First, we present a thorough characterization of storage offers relying on traffic traces collected in an operational network for one month. Thanks to the dataset at our availability, we detail the typical use of three cloud storage services, i.e., Dropbox, Google Drive and Microsoft OneDrive, highlighting the differences in access and usage patterns depending on the service considered and on the device used. Second, we propose a generic benchmarking methodology to compare several services using active network measurements and implement tools to automate the benchmarks. We analyze the performance of 11 storage services under various synchronization scenarios, shedding light on the impact of design choices on performance. Realistic testing conditions are applied leveraging the findings of usage characterization, thus defining workloads inspired by real-life operations.

Our contributions can be summarized as follows:

- We characterize the usage of three popular cloud storage services through the analysis of traffic collected in an operational ISP network and representative of the actual behavior of 20,000 residential customers;
- we study service popularity, connection frequency, data volume users exchange with cloud storage services, and assess the achievable performance given the availability of broadband connections;
- we shed light on differences in usage patterns and performance according to the terminal type (e.g., PC client, mobile App, or Web interface) being used to access the service;
- we introduce a methodology through which active experiments can be carried out and helps to determine client capabilities and data center locations, as well as to benchmark cloud storage;

- we apply such methodology to identify the capabilities implemented and contrast the performance offered by 11 cloud storage providers adopting a black-box testing approach, i.e., performance metrics are obtained without instrumenting clients. Workloads are either tailored to verify the availability of specific client capabilities or inspired by the outcomes of the usage characterization mentioned before;

- we document how different providers implement cloud storage services focusing on client capabilities and system architecture, and highlight how design choices can sensibly affect performance.

This part of the thesis is organized as follows:

Chap. 3 presents the passive characterization of cloud storage usage. The dataset collected and processed for the purpose is detailed in Sec. 3.1. Results are presented in Sec. 3.2, which provides statistics on connection frequency and on the volume of data exchanged with cloud services, and in Sec. 3.3, which reports on the achievable performance in terms of throughput. Finally, Sec. 3.4 details the differences in usage and performance depending on the type of terminal used.

Chap. 4, instead, describes the techniques and the results for active experimentations. Specifically, the methodology and the testbed designed to execute experiments are described in Sec. 4.1. An investigation on advanced client capabilities is reported in Sec. 4.2, system designs and protocols are discussed in Sec. 4.3, while performance results for the complete synchronization chain are illustrated in Sec. 4.4.

Chapter 3

Personal Cloud Storage

Usage Patterns Characterization

In this chapter, we characterize the typical usage of three popular cloud storage services, namely Dropbox, Google Drive, and Microsoft OneDrive. We leverage a large passive measurement campaign covering a month of traffic captured in an operational residential network connecting approximately 20,000 households. The statistics here presented are thus representative of the actual usage of personal cloud storage by end users.

Several aspects of cloud storage usage are tackled, spanning from the popularity of cloud services to the connection frequency of users. In addition, we consider the amount of traffic produced by each user and the overall traffic exchanged with the cloud to assess the quantity of resources demanded to the network. We also focus on the achievable performance in both the upload and the download directions, identifying potential bottlenecks limiting the transfer rate and not allowing users to saturate the high-speed access networks offered today by ISPs.

We complete the picture by identifying the implications of different terminal types on usage patterns and performance. Users, indeed, can access files stored on the cloud through PC clients, from Apps running on smartphones and tablets, or via the Web interface.

3.1 Methodology and Dataset

We rely on passive measurements collected in operational networks to characterize how people use different cloud services and the influence of terminals on both usage and performance. We monitor the PoP of an ISP where more than 20,000 residential households are connected and we leverage the flow-

Table 3.1: FQDNs for storage provider and device identification.

Provider	Control	Upload	Download
Dropbox	client*.dropbox.com client-lb.dropbox.com notify*.dropbox.com	block.dropbox.com dl-balancer.*.dropbox.com[1] api-content.dropbox.com[2] dl-web.dropbox.com[3]	block.dropbox.com dl-client.*.dropbox.com[1] api-content.dropbox.com[2] dl-web.dropbox.com[3]
OneDrive	login.live.com onedrive.live.com	*.storage.live.com	*.livefilestore.com
Google Drive	account.google.com client*.google.com	upload.drive.google.com upload.docs.drive.google.com	googledrive.com *-docs.googleusercontent.com

[1] Access from synchronization clients running on PCs.

[2] Access from mobile devices.

[3] Access from the Web interface.

level summaries exported by Tstat (see Sec. 1.2.1). Households access the Internet via Asymmetric Digital Subscriber Line (ADSL) or Fiber to the Home (FTTH) connections and use PCs, smartphones, and tablets at home. FTTH offers up to 100 Mb/s and 10 Mb/s in downstream and upstream, respectively. Conversely, ADSL offers up to 24 Mb/s and 1 Mb/s.

Among the many measurements provided by Tstat, we base our work primarily on (i) the client IP address; (ii) the total number of bytes exchanged with servers; (iii) the timestamp of the first and the last packet with payload in the flow; and (iv) the FQDN the client resolved via DNS queries prior to opening the flow. More specifically, we use the client IP as identifier for the household thanks to the static IP address allocation in place at the ISP, and the FQDN to identify the cloud storage service being contacted. The number of bytes and the timestamps are instead used to assess the amount of traffic exchanged with each cloud provider and compute performance figures.

We focus on three among the most popular personal cloud storage solutions: Dropbox, Google Drive, and OneDrive. To isolate the traffic generated by these applications, we build a list of FQDNs used by cloud servers. To compile such list, we run toy experiments in a controlled environment where we monitor the DNS requests and define a match between each FQDN contacted and task it is called to accomplish.

Next, we use this list to filter the records exported by Tstat. For instance, *upload.drive.google.com* isolates the traffic sent from users to Google Drive servers, whereas **.storage.live.com* is used when contacting OneDrive. In some cases, it is possible to distinguish traffic generated by different devices used by customers to access the cloud. For instance, *dl-client*.dropbox.com* identifies traffic related to the Dropbox application running on PCs, while

Table 3.2: Overview of our dataset.

Provider	Households	Upload (GB)	Download (GB)
Dropbox	5,020	681	1,209
OneDrive	4,910	336	132
Google Drive	2,016	164	311
Total	7,349	1,181	1,652

api-content.dropbox.com is used by smartphones and tablets applications. The complete FQDNs list is reported in Tab. 3.1.

3.1.1 Dataset Description

In the following, we focus on the traffic observed during the whole month of October 2014. Tab. 3.2 summarizes our dataset. It shows the number of households running each service, and the total amount of exchanged traffic. Dropbox is the most popular service, with $\approx$ 1.9 TB transferred in total. OneDrive is as popular as Dropbox, but the amount of traffic exchanged with cloud servers is much smaller, especially considering the download. Google Drive comes third, with 10% of users having used it at least once in a month. Some households use more than one cloud service. The total number of *unique* households is $\approx$ 7,300 (36%). Overall, we observe 1.1 TB of uploads and 1.6 TB of downloads related to cloud storage.

3.2 Characterization of Cloud Storage Usage

We study whether users of different cloud providers present different behaviors. We show that users exhibit distinct patterns across providers. Indeed, Dropbox population is more heterogeneous, connecting less, but exchanging more data than Google Drive users. OneDrive users, in contrast, are frequently on-line, but hardly exchanging any data.

3.2.1 Connection Frequency

We first investigate if users of Dropbox, OneDrive and Google Drive are equally active. Fig. 3.1 shows how frequently users access storage services. Left plot shows the cumulative population of unique households for each service. If we consider one day (leftmost point), we see about 1,500 unique households for Google Drive, 2,400 for Dropbox, and 3,000 for OneDrive. Moving to the right, the number of unique households quickly increases in

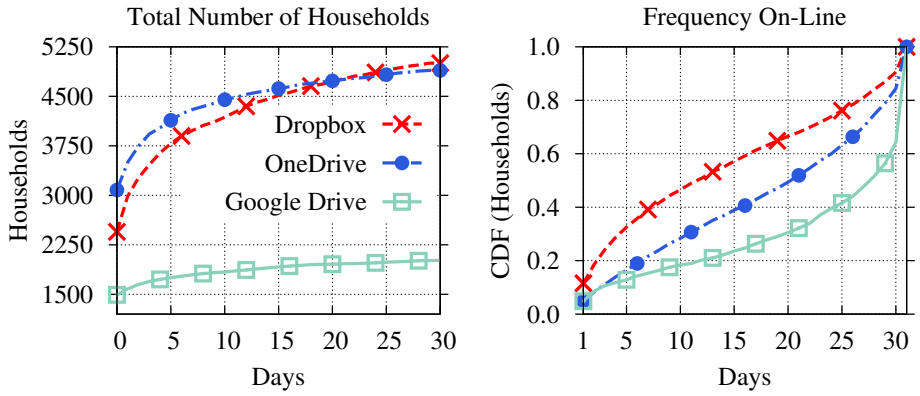


Figure 3.1: Number of households using cloud storage services and the frequency they connect. Dropbox and OneDrive lead in number of households, but Google Drive users are steadily connected.

the first 5–10 days, after which the growth rate gets smaller. Significant differences in population sizes are visible – e.g., Dropbox and OneDrive are used in around 25% of the 20,000 monitored households (see Tab. 3.2). Compared to 2012 [15], Dropbox penetration more than doubled, but competitors are closer now – e.g., OneDrive and Google Drive were far below 1% in 2012. Interestingly, Dropbox and OneDrive curves of unique households intersect in the plot. This suggests that OneDrive users access the service using PC applications that contact the service as soon as they bootstrap the PC. For Dropbox there are instead some occasional users that access the service from other interfaces, e.g., clicking on direct Web download links.

To investigate it further, we focus on how many times users access each cloud service. Right plot in Fig. 3.1 depicts the empirical Cumulative Distribution Function (CDF) of households with respect to the number of days they connect to the cloud provider. Distinct patterns emerge. For Dropbox, the fraction of households occasionally accessing the service is higher. For instance, more than 12% of the households access the service for one day only, while 55% use it for 15 or less days in a month. For OneDrive, these percentages decrease to 4% and 40%, respectively, showing a more frequent access pattern, likely driven by the fact that OneDrive is integrated into recent versions of Microsoft Windows. This results in having a PC or a smartphone contacting OneDrive servers as soon as it is connected to the network. For Google Drive, users connect even more frequently: 60% of the households have accessed servers in 25 out of the 31 days, and 40% access the

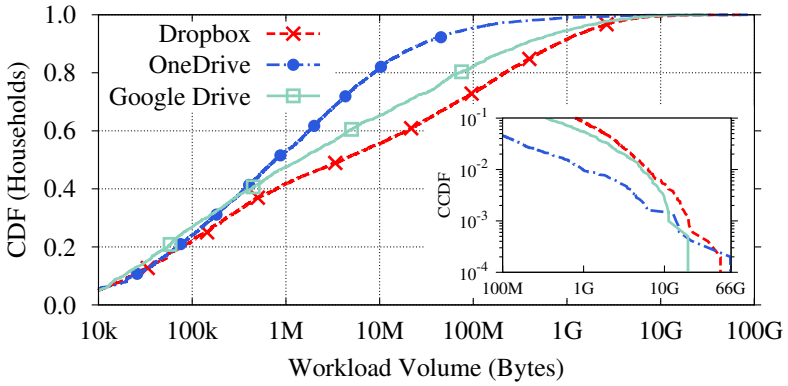


Figure 3.2: Total storage workload per household. Dropbox exchanges more data when present in a household. Google Drive comes close. OneDrive seldom synchronizes large volumes.

service on a daily basis. This is explained by the fact that Android devices automatically connect to Google Drive servers for backups.

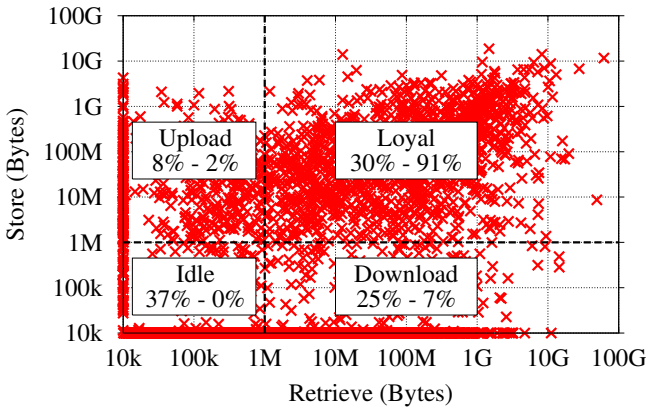
Results allow us to conclude that users of each cloud service have specific dynamics, with Google Drive ones accessing it more often, even if Dropbox and OneDrive appear to have a higher market presence. Intuitively, these patterns would also reflect in the usage of the services.

3.2.2 Data Volumes

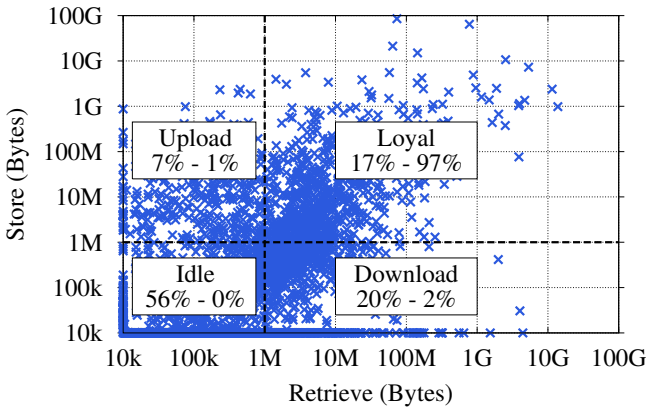
We now compare users in terms of generated traffic. Fig. 3.2 provides details of the storage workload each household exchanged in the whole month.

First, about 40–50% of the households generate very little traffic (less than 1 MB). These are *idle* users. Considering households that generate significant workload, we observe that Dropbox users clearly exchange much more data than others. Indeed, 30% of the households using Dropbox exchange more than 100 MB in a month versus only 4% of OneDrive users. Google Drive is in between. The distribution tail shown in the Complementary CDF (CCDF) in the inset highlights that 1% of Dropbox users exchange 8 GB or more.

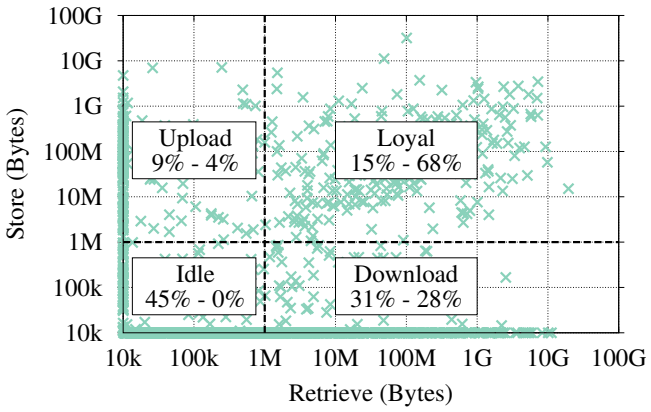
As clearly shown, the vast majority of OneDrive users seldom exchanges any data, reinforcing the intuition that a relevant amount of OneDrive users are actually Windows users accessing the service, but not using it. Note the existence of *loyal* OneDrive users in the CCDF, where few households exchanged more than 60 GB in the whole month. Google Drive users usually exchange less traffic than Dropbox, even though they access the service almost every day.



(a) Dropbox



(b) OneDrive



(c) Google Drive

Figure 3.3: Volume stored and retrieved by each household (in log axes). Households are divided in groups with limits at 1 MB. Labels report the percentage of households and bytes per group. Many idle households are seen in OneDrive and Google Drive.

3.2.3 Usage Scenarios

Both Dropbox and Google Drive users generally download more from the cloud than what they upload (respective ratio 1.8 and 1.9 – see Tab. 3.2). OneDrive users show the opposite pattern: 2.5 times more upload volume than download. We investigate this difference in more details.

Fig. 3.3 provides a finer view into the storage workload by characterizing the total data stored and retrieved per household in the complete month. Each dot in the figure represents a household. By setting a threshold at 1 MB, we identify four behaviors: (i) *Idle*, i.e., households with little upload and download traffic; (ii) *upload* only; (iii) *download* only; or (iv) *loyal*, i.e., household with large amounts of data for both download and upload. Households with less than 10 kB are placed on the axes. Numbers in Fig. 3.3 report the percentage of households and bytes in each group, respectively.

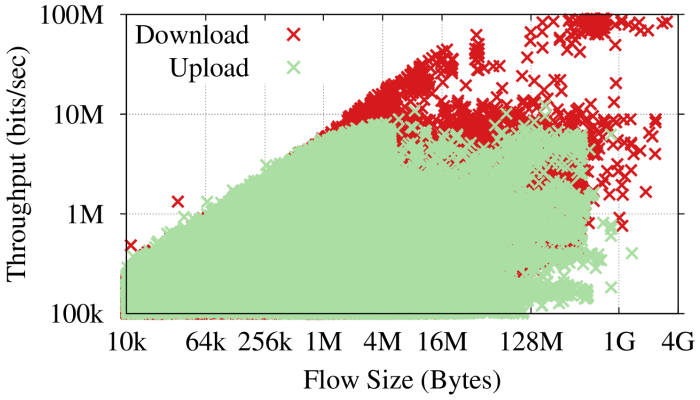
Focus first on Dropbox results. We see that a large percentage (37%) of households is idle and exchanges a negligible amount of data in one month. 91% of the workload is produced by loyal users (30% in population). Interestingly, the amount of downloaders (25%) is much higher than uploaders (8%). Thus, Dropbox users download more than what they upload, possibly to synchronize multiple PCs or to retrieve content shared by other users.

Focus now on OneDrive. More than half (56%) of the households are idle, while only around 17% are loyal. The latter generates 97% of the overall volume, with few households uploading close to 100 GB, hence biasing usage towards more upload than download. We conjecture these households are using the service to perform large backups.

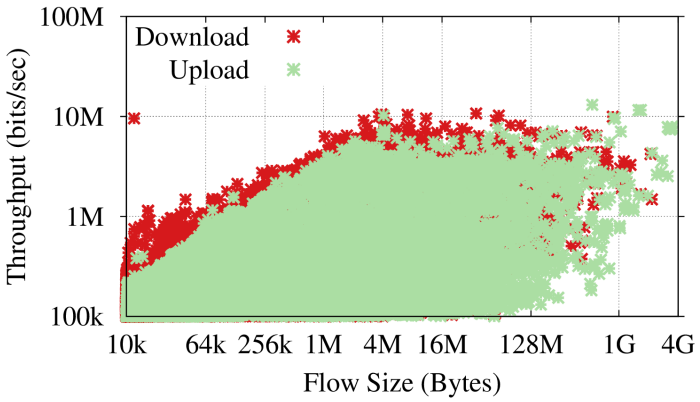
Finally, observe Google Drive: 45% of the households are idle. The percentage is higher than Dropbox likely because of Android users, which make little usage of Google Drive, but still connect to it on background. The fraction of loyal users (15%) is similar to OneDrive, but they exchange less data. 31% of the households fall in the download-only class. They tend to download a lot, generating 28% of total volume.

3.3 Performance

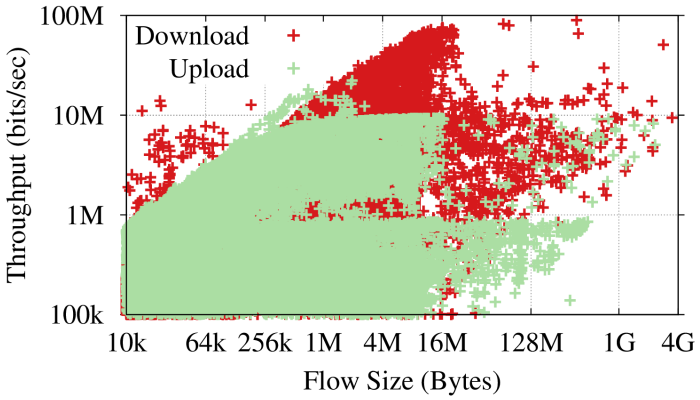
We focus on performance, taking throughput as the main index, and contrasting it across both services and terminals. We observe several bottlenecks that impair performance: (i) TCP slow-start dominates short transfers; (ii) services (e.g., OneDrive) artificially limit the throughput; and (iii) protocol designs (per client terminal) have a great impact.



(a) Dropbox - All



(b) OneDrive



(c) Google Drive

Figure 3.4: Throughput for different services. Dropbox can saturate the line capacity, while OneDrive shapes throughput. Complex client policies limit Google Drive throughput for large workloads.

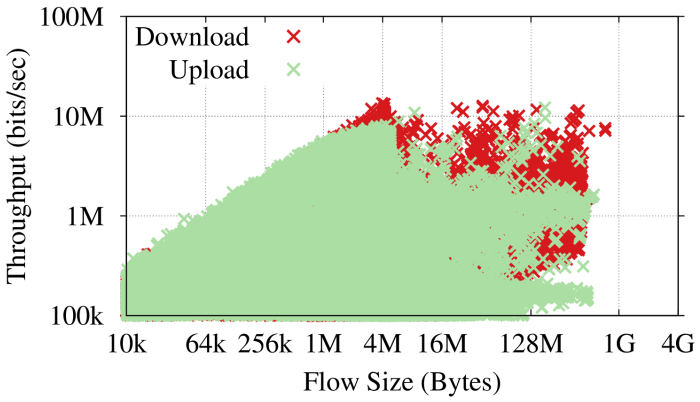
3.3.1 Performance of Storage Clients

Given a TCP flow, we consider the size of the application payload, and the timestamp of the first and the last data packet to calculate the flow duration. We then define the throughput as the ratio between payload size and flow duration. Fig. 3.4 shows the results. For each flow, we plot throughput (y-axes) versus flow size (x-axes). Red and green markers are used in the figure for download and upload, respectively. Dropbox, OneDrive, and Google Drive are shown from top to bottom. Plots are in log scale.

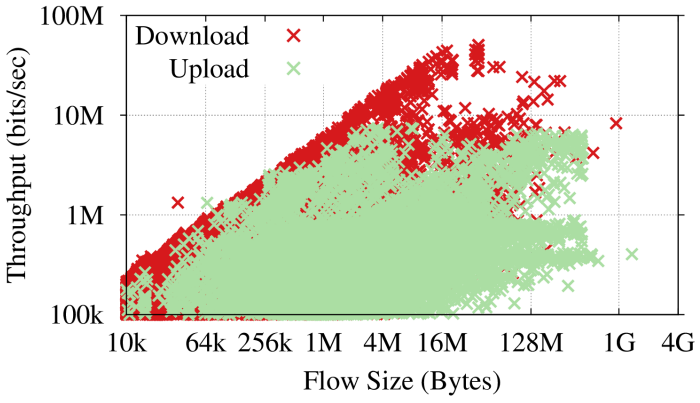
First, notice the linear dependence between the maximum achieved throughput and flow size for small transfers. This is due to TCP slow-start, which requires several Round Trip Times (RTTs) to end. The transfer of short workloads ends before TCP can grow the congestion window, thus without saturating the path capacity. In such cases, latency dominates the throughput. Interestingly, both Dropbox and OneDrive have data centers in the U.S. and the RTT to our vantage point is higher than 100 ms. Google Drive data center is instead much closer (RTT=15 ms). Notice how the maximum throughput for Google Drive is higher than others when committing short workloads. Yet, short transfers face the TCP bottleneck.

Second, focus on large transfers ($\geq$ 1 MB for uploads and $\geq$ 10 MB for downloads). Here TCP reaches steady state and can saturate the available capacity. The upload rate limits of 1 Mb/s (ADSL customers) or 10 Mb/s (FTTH customers) are clearly visible. For downloads, both Dropbox and Google Drive allow customers to saturate their downlink capacity at $\approx$ 20 Mb/s (100 Mb/s) for ADSL (FTTH) customers in some occasions. On the contrary, OneDrive artificially forces the download throughput to less than 10 Mb/s.

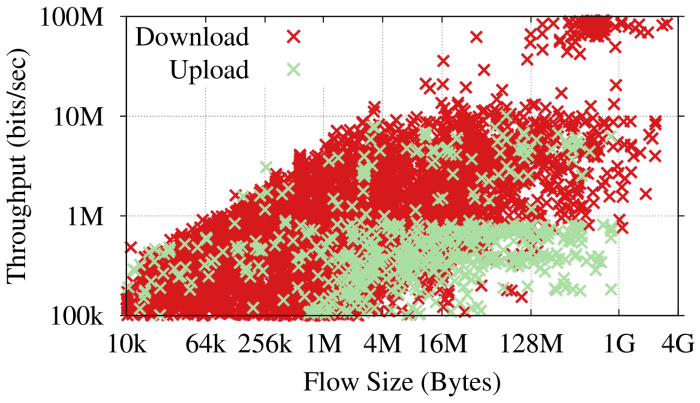
At last, Dropbox and Google Drive performance is affected by complex client policies. For example, note the drop in the number of flows above 16 MB for Google Drive. This is due to the fact that large files are exchanged in chunks not exceeding 16 MB (see Sec. 4.2 for more details) and each chunk is sent over an independent TCP flow. This policy results in a strongly limited download rate for files larger than the chunking threshold.



(a) Dropbox PC



(b) Dropbox Mobile



(c) Dropbox Web

Figure 3.5: Throughput for different client terminals. Complex patterns appear for Dropbox from PC clients, limiting both throughput and flow size. The line capacity is saturated only through mobile apps and Web interface. Notice how Web access is used for large downloads.

3.3.2 Performance per Terminal Type

Plots in Fig. 3.5 split Dropbox results per client terminal, depicting the performance of Dropbox according to the type of device used to access the service. We split the data shown in Fig. 3.4(a) into three categories: PC, mobile and Web. The latter includes both the Dropbox main Web interface and direct links sent among users to share files.

Two major considerations hold. First, focusing on Dropbox used through PC clients (Fig. 3.5(a)), no single flow is able to saturate the FTTH line capacity, especially for downloads. We notice a clear correlation between flow size and the achieved performance. Also Dropbox, similarly to Google Drive, splits large files into chunks of 4 MB (see Sec. 4.2). When files larger than 4 MB are to be transferred, the PC client slices them into chunks and may send chunks over independent flows. When the flow size exceeds this threshold, the larger the flow is, the lower is the maximum achieved throughput. This can be explained by a combination of factors, including the complex operations performed by the client to optimize network usage, the delay introduced by commit operations when dealing with the several chunks that compose large file sets, etc. At last, flows are limited to around 400 MB by design, which penalizes the transfer of workloads larger than that.

Second, notice in Figs. 3.5(b) and 3.5(c) how the bottlenecks identified at the PC client are no longer present in mobile and Web accesses. We see in particular that the Web interface is the only one able to take full advantage of the high-speed lines at households' disposal. This is explained by the mechanisms adopted to download large files from the Web interface. Clients perform a simple HTTPS download that reaches up to 100 Mb/s on FTTH connections. Note also the difference in the amount of downloads and uploads among the client terminals, better detailed later.

Although cloud storage presents high network demands, artificial shaping policies, client design choices, and terminal limits can affect performance.

3.4 Usage per Client Terminal

We focus on Dropbox since we identify a significant number of households accessing the service from different terminals. Our findings suggest that usage patterns change considerably according to the type of terminal. Uploads from mobile devices are common and usually larger than those from PC. On the other hand, downloads are rare and small. At last, the Web interface is hardly used to upload content, being instead preferred for downloads.

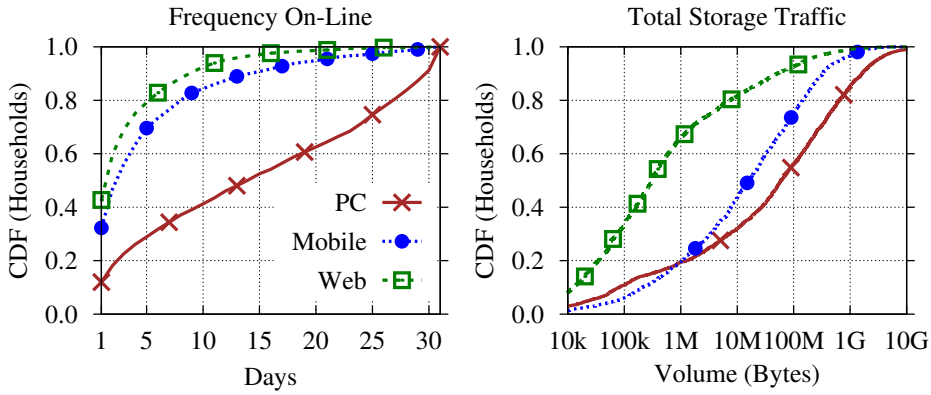


Figure 3.6: CDF of households active with each type of terminal, and their total traffic. Web and mobile clients are rarely active. However, mobile clients produce lots of traffic.

3.4.1 Connection Frequency and Volume

We evaluate how often households using Dropbox connect from each type of terminal and how much data they exchange. We identify 2,196, 1,628 and 3,832 households using PC, mobile and Web, respectively. For Web, 3,581 households use it to consume direct links sent by other users to share files.

The left plot of Fig. 3.6 depicts the CDF of the number of days in which households interacting with each type of client transfer data. Focusing on the leftmost point in the figure, note how 40% of the households using the Web interface connect only once in a full month. This is expected given the direct link usage scenario. Some of these households do not even have Dropbox installed on PCs. Still, they generate workload when receiving Web links from others. Similarly, 30% of the households with mobile clients connect only once in a month. Compare it to PC clients, in which 90% (60%) of the households exchange data in more than 1 day (10 days).

To explore this further, the right plot in Fig. 3.6 depicts the CDF of households with respect to the volume they exchange for different terminals. We see that the Web interface creates the lowest amount of traffic. Only 20% of the households transfers more than 10 MB in a month. On the other extreme, PC clients generate more traffic than others – e.g., 70% exchanges more than 10 MB. Much more interesting is the behavior for mobile terminals. Although households rarely use Dropbox from mobile devices (see left-hand plot), the amount of traffic from such clients is noteworthy: 20% (50%) of households exchanges more than 100 MB (10 MB).

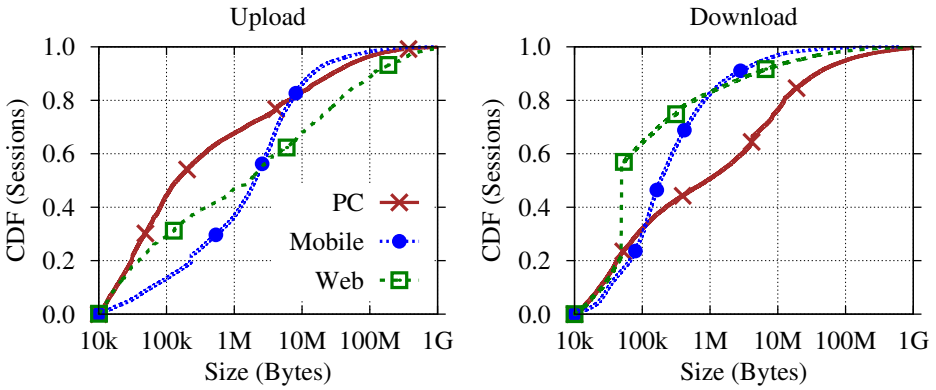


Figure 3.7: Transferred bytes per download or upload session. Most PC uploads are short, while downloads are the largest ones. Mobile uploads instead are large, while mobile downloads are rare and short.

3.4.2 Workloads

Overall, users uploaded 462 GB, 180 GB and 38 GB from PC, mobile and Web, respectively. Here we highlight a major change in cloud storage usage habits. Whereas upload in mobile was marginal in 2012 [15], it is now equivalent to 37% of PC upload volume. In terms of downloads, PCs still dominate, peaking at 980 GB, while only 51 GB were downloaded on mobile terminals, and 177 GB from the Web during the entire month.

We dissect these figures by evaluating the size of workloads people create *at once*. We again split flows into upload or download. Flows on PCs are however biased towards 4 MB due to the internal mechanisms adopted by the Dropbox client. We overcome the bias by evaluating storage *sessions*: Whenever upload or download flows in the same household overlap in time, we merge them into a single storage session. We then analyze the size of storage sessions per device type.

Fig. 3.7 depicts the size of upload (left-hand side) and download sessions (right-hand side). Note how most of them on PCs are very small, 70% carrying less than 1 MB. This behavior has been associated [15] to the dominance of small files as well as to the capabilities Dropbox implements to optimize network usage, such as compression and delta encoding. Compare now with the line depicting mobile sessions. Mobile uploads are concentrated between 1 MB and 10 MB. We conjecture this behavior is associated to the usage of cloud storage to store pictures and videos captured with mobile devices. Web upload does not implement smart policies and thus workloads grow smoothly with no clear bias towards specific sizes.

Comparing left and right plots, we see that downloads are larger than uploads on PCs. Again, this behavior has been associated to the large workloads that are spread to many devices in a household. In particular, at boot time PCs often have to download full batches of files, modified or added elsewhere. In contrast, uploads are sent as soon as changes occur. Mobile downloads are much rarer and smaller than uploads. This is explained by the fact that mobile clients do not synchronize content unless users explicitly request it. The CDF for Web is biased towards 50 kB, which is the typical size of picture previews on the Dropbox website.

All in all, we conclude mobile terminals are mostly content producers, Web interfaces are often used to fetch content via direct links, whereas PCs dominate cloud storage in terms of data volumes for file synchronization.

3.5 Conclusions

This chapter evaluated personal cloud storage using passive measurements collected in an operational network. Whereas Dropbox in PCs used to dominate the market [15], now the landscape is changing. Usage of cloud storage is increasing both in competitors, such as OneDrive and Google Drive, as well as in different terminal types, like mobile devices.

This study on typical usage and performance yielded new insights. For example, we observed that usage across providers is distinct, with high numbers of OneDrive and Google Drive users that are active without creating much workload. This is a consequence of the close integration of cloud storage into modern OSes. Performance bottlenecks were highlighted, including explicit traffic shaping policies and the impact of well-known TCP bottlenecks on small workloads. Finally, we noticed a large increase in traffic from mobile terminals, which seems to be related to multimedia content produced by tablets and smartphones.

While measurements in this work are tied to the vantage point used, we believe they provide valuable information into overall trends and are of interest to understand and track the evolution of personal cloud storage systems. In addition, collected statistics on exchanged bytes constitute a solid base to define realistic file sets to be used as workload for active experiments and thus evaluate performance of cloud storage services in real-file scenarios.

Chapter 4

Personal Cloud Storage

Benchmarks and Comparison

Despite the high public interest in personal cloud storage services, little information about system design and actual implications on performance is available. Systematic benchmarks to assist in comparing services and understanding the effects of design choices are still lacking. To fill such gap, this chapter proposes a methodology to study personal cloud storage services through active and customizable experiments. We design a benchmarking environment to run realistic and repeatable tests so to unveil service architectures and capabilities. By generating specific workloads and testing cloud services under different scenarios, we assess the performance offered to end users, helping in the search for the service providing the demanded features.

The effectiveness of the methodology is verified in a case study involving 11 services, including 9 popular cloud providers and 2 private cloud installations. The results presented are achieved through continuous and automated measurements and highlight, among the other results, service availability on longer time spans and verify the consistency of offered performance.

Our case study targets two major goals. Firstly, it sheds light on how providers tackle the problem of synchronizing people's files, revealing differences in client software, capabilities, synchronization protocols and data center placement policies. Secondly, we evaluate how such choices impact end user performance. To this end, we perform a large series of experiments using different workloads, taking the perspective of customers connected from a single location in Europe. We contrast the measured performance against the characteristics of each service, thus highlighting the consequences of several design choices for both end users and the public Internet network.

4.1 Measurement Methodology

This section describes our methodology to study cloud storage services. The methodology is developed around an active testbed that allows us to run specific benchmarks. First, we aim at (i) testing client capabilities, and (ii) highlighting protocol design choices and data center placement decisions. Then, the testbed is used to measure the implications of these two aspects on performance under different workloads.

4.1.1 Goals

In the design of our methodology, we follow the traditional black-box testing approach. We consider an *application-under-test* that is run in a controlled environment, where we can impose an input sequence while recording the external behavior of the system. We instrument a testbed in which one or more *test computers* run the desired application-under-test. A *testing application* generates a pre-defined workload, i.e., it creates specific file(s) that the cloud storage service should synchronize among the test computers. At the same time, the testbed passively collects the traffic exchanged in the network to obtain a trace of the execution.

We assume to have control neither on the application-under-test nor on the network. We want tests to be repeatable and automatically performed, both to save time and to obtain average estimations. Moreover, since we target the comparison of tens of different services, we need the whole testing methodology to be able to run without constant supervision, post-processing and analyzing recorded traces automatically.

4.1.2 Testbed and Tools

Fig. 4.1 depicts our testbed. Its core component is the testing application that orchestrates experiments and records network traffic. Two or more test computers run the application-under-test. For simplicity, we consider only two test computers here, although our methodology is generic and supports multiple clients as well. Our testing application receives benchmarking parameters describing the sequence of operations to be performed (step 0 in Fig. 4.1). Then, the testing application acts remotely on *Test Computer 1* (step 1) by means of a File Transfer Protocol (FTP) server, generating workloads in the form of file batches. Once the application-under-test detects that files have changed, it starts to synchronize them to the cloud (step 2). The application-under-test running on *Test Computer 2* detects modifica-

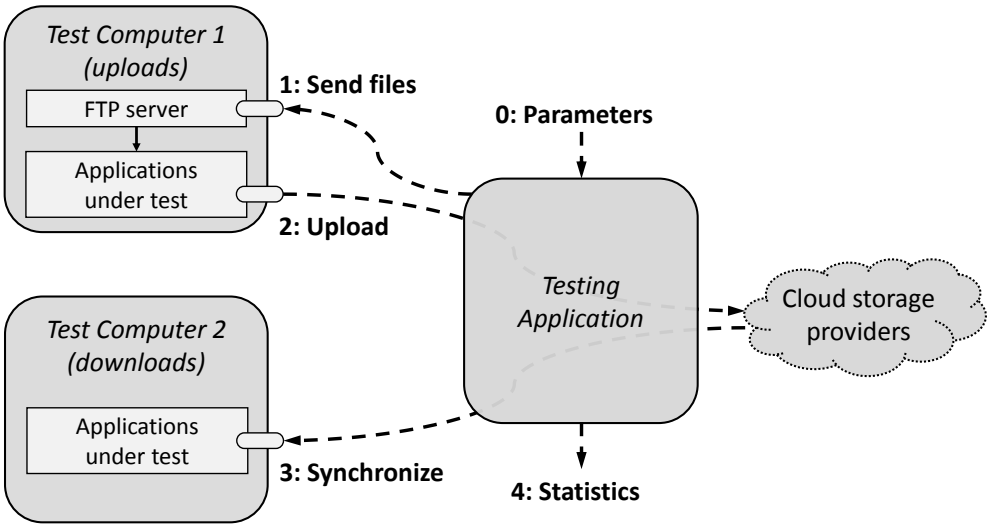


Figure 4.1: Testbed and workflow of the benchmarks.

tions and downloads the new content (step 3). Exchanged traffic is recorded during all steps and processed to compute performance metrics (step 4).

We setup the testbed using a single Linux server connected via a 1 Gb/s Ethernet card to the Politecnico di Torino campus network, in which Internet connectivity is offered by a 10 Gb/s link. The server controls the experiments by running the testing application, and hosts two virtual machines that run the test computers (Windows 7 Enterprise). Both test computers have their network interfaces connected to a virtual network, while the Linux server is instrumented to act as a router and provide Internet connectivity to the virtual network. This setup allows the server to easily observe traffic exchanged with the test computers. We tested also physical machines without noticing any difference in results. However, care must be taken in calibrating the virtual environment – e.g., to avoid network shaping parameters, dimensioning the host server, etc.

4.1.3 Client Capabilities and Storage Protocols

We first study protocols used to communicate with the cloud. A manual inspection on packet traces reveals that most cloud storage services adopt HTTPS – i.e., they encrypt payload. Only in few cases, some control information is exchanged by means of non-encrypted protocols such as HTTP. This is commendable given the privacy issues that could arise from transporting user files without encryption. However, the adoption of encrypted

protocols complicates the understanding of the operations performed by the application-under-test.

Next, we aim at understanding whether services implement any advanced capabilities to manipulate files on the cloud. In particular, previous work [15] showed that storage services can implement client capabilities to optimize network usage and speed up transfers. These capabilities include: (i) *Chunking* – i.e., splitting large files into a maximum size data unit; (ii) *Bundling* – i.e., the transmission of multiple small files as a single object; (iii) *Deduplication* – i.e., avoiding re-transmitting content already available on servers; (iv) *Delta encoding* – i.e., transmitting only modified portions of files; (v) *Compression* – i.e., compressing files before transmission; and (vi) *P2P synchronization* – i.e., exchange files among devices without involving storage servers.

For each case, a test has been designed to observe if the given capability is implemented. We describe these tests in Sec. 4.2. Intuitively, our testing application generates specific file batches that would benefit from the availability of a capability. The exchanged traffic is analyzed to observe if these benefits are present. Finally, we note that these experiments only need to be executed when a new client version of a cloud storage service is released.

4.1.4 Data Center Locations

The data center locations and data center ownership are important aspects of cloud storage services, having both legal and performance implications. To identify how services operate, we leverage the IP address and hostname of servers contacted by an application-under-test when (i) it is started; (ii) files are manipulated; and (iii) it is in idle state.

In cloud services, different IP addresses can be returned when querying the DNS. Load-balancing techniques are often in place to split the workload based on the client location [66]. Following an approach similar to [67], we resolve all hostnames using more than 2,000 open DNS resolvers spread around the world. The list of resolvers has been manually compiled from various sources and covers more than 100 countries and 500 ISPs. This methodology allows us to create a list of server IP addresses used by a cloud storage provider.

Once the list of IP addresses is obtained, we use the `whois` service to verify their ownership, i.e., the name of the company owning the IP addresses. Next, we look for the geographic location of servers. Since popular geolocation databases have serious limitations regarding cloud providers [68], we rely on a hybrid methodology that makes use of: (i) Informative strings (i.e., International Airport Codes) revealed in the hostname or by reverse DNS lookups; (ii) the shortest RTT to PlanetLab nodes [69]; and (iii) active

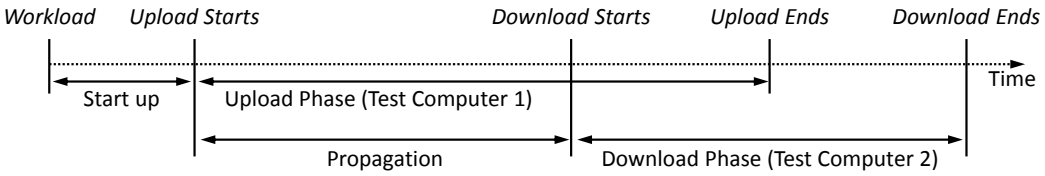


Figure 4.2: Typical synchronization cycle in a personal cloud storage service.

`traceroute` to spot the closest well-known location of a router. Previous work [66, 70] indicates that these methods provide an estimation with about a hundred of kilometers of precision. Since we aim at a coarse reference of the location from where cloud providers operate (e.g., at country level), this estimation is sufficient for our goals.

4.1.5 Benchmarking Performance

After knowing design choices of services, we use our testbed to check their influence on performance. We engineer a methodology to calculate metrics related to typical phases of the synchronization cycle as depicted in Fig. 4.2.

A workload is generated by the testing application based on a benchmark definition. A variable number of files with different content and size is created and manipulated, e.g., text files composed of random words from a dictionary, images with random pixels or random binary files. In addition, a percentage of file replicas can be specified to test how the service reacts when synchronizing repetitive content.

Performance metrics are calculated. Fig. 4.2 depicts the typical steps while synchronizing a file batch. We calculate: (i) The duration of the silent period before Test Computer 1 reacts to a new workload (i.e., start up); (ii) the duration and the amount of traffic while Test Computer 1 uploads files; (iii) the delay between the start of the upload and the download (i.e., propagation time); and (iv) the duration and the amount of traffic while files are downloaded from Test Computer 2.

Each phase is marked with a timestamp identified from network events based on the (previously learned) behavior of each application-under-test. Since most cloud storage services are proprietary and use encrypted protocols, determining such events from network traces is challenging. We learn the typical behavior of each service by means of controlled experiments and manual inspection, following a typical trial-and-check process. Once significant events are identified and validated, they are coded in (application-specific) post-processing scripts.

Table 4.1: Considered cloud storage services. The list includes 11 services, among which 2 are run privately.

	Service	Version
Public	Box	4.0.5101
	Cloud Drive (Amazon)	2.4.2013.3290
	Copy (Barracuda)	1.45.0363
	Dropbox	2.8.4
	Google Drive	1.16.7009.9618
	hubiC (OVH.com)	1.2.4.79
	Mega	1.0.5
	OneDrive (Microsoft)	17.3.1166.0618
	Wuala (LaCie)	*Olympus*
Private	Horizon (VMware)	1.5.0-12211212
	ownCloud	1.5.2

4.1.6 Storage Services Under Test

We consider overall 11 services under study, all being the latest version at the time of writing. We restrict our analysis to native clients since this is the largely preferred means to use cloud storage services [15].

Tab. 4.1 lists the considered services. These include: (i) Popular players (e.g., Dropbox, Google Drive, Microsoft OneDrive, Amazon Cloud Drive); (ii) services with a strong presence in Europe, whose data centers are close to our testbed location (e.g., hubiC, Wuala and Mega); and (iii) private cloud services that are run in our institution (i.e., ownCloud and Horizon). Such variety allows us to compare the impact of multiple protocol design choices, client capabilities, and data center placement policies on cloud storage performance. We only consider services offering free storage space, and those operating in a cloud infrastructure. Therefore, services such as SugarSync and BitTorrent Sync are not evaluated.

Finally, we note that Wuala is the only service encrypting files on the local computer before transferring them to the cloud. This can have extra processing costs and impair performances, as we will check in coming sections.

4.2 Client Capabilities

We first evaluate which capabilities are implemented by different services. Findings are then summarized in Tab. 4.2.

Table 4.2: Summary of the capabilities implemented in each service.

Service	Bundling	Chunking	Compression	Dedupl.	Delta Encoding	P2P Sync
Box	no (*mc*)	no	never	no	no	no
Cloud Drive	no (⇑*mc*, ⇓*mt*)	no	never	no	no	no
Copy	no (*mt*)	5 MB	never	**yes**	no	**yes**
Dropbox	**yes**	4 MB	always	**yes**	**yes**	**yes**
Google Drive	no (*mc*)	8 MB (⇓only)	**smart**	no	no	no
hubiC	no (*mt*)	no	never	no	no	no
Mega	no (*mc*)	1 MB	never	partially	no	no
OneDrive	no (*mt*)	⇑4 MB, ⇓1 MB	never	no	no	no
Wuala	no (*mt*)	4 MB	never	**yes**	no	no
Horizon	no (*mt*)	1 MB (⇑only)	never	partially	no	no
ownCloud	no (*mt*)	no	smart (⇓only)	no	no	no

mc: One or multiple TCP connections per file;

mt: One or multiple application layer transactions per file.

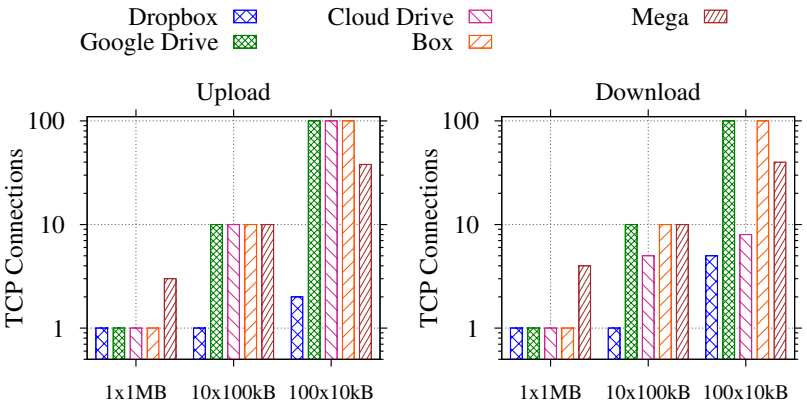
⇑ and ⇓ refer to the upload and the download experiments, respectively.

4.2.1 Bundling

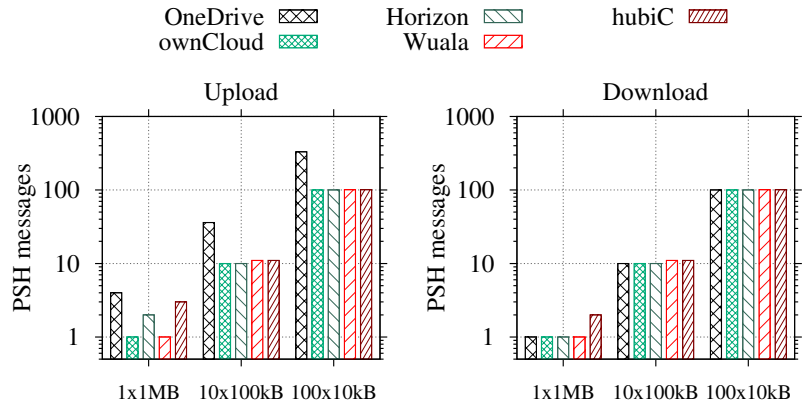
When a batch of files is transferred, files could be bundled, so that transmission latency and control overhead are reduced. Our experiment to check how services handle batches of files consists of 3 file sets of 1 MB: (i) 1 file of 1 MB; (ii) 10 files of 100 kB; and (iii) 100 files of 10 kB. We then estimate the number of upload and download operations.

These experiments reveal three design trends. Results are reported in Fig. 4.3, in which different plots are shown for uploads and downloads. Some services are omitted from plots in this section to help visualization. A first group of services (Box, Cloud Drive, Google Drive and Mega) opens several TCP connections when submitting multiple files. Notice in Fig. 4.3(a) how the number of TCP connections opened by Box and Google Drive is exactly the same as the number of files. Compare to Dropbox, which instead uses only few TCP connections to transfer multiple files. The first is a strong indication that the service is managing each file in a separate transaction. Notice that when several connections are used, the traffic overhead can be significant, owing to TCP and TLS/SSL negotiations.

Mega and Cloud Drive sometimes group files into fewer TCP connections (see download plot). This is also the case for other services – i.e., Copy, Horizon, hubiC, OneDrive, ownCloud and Wuala. However, they submit files in separate transactions. Fig. 4.3(b) illustrates this design, which can be inferred from the bursts of packets delimited by TCP segments in which



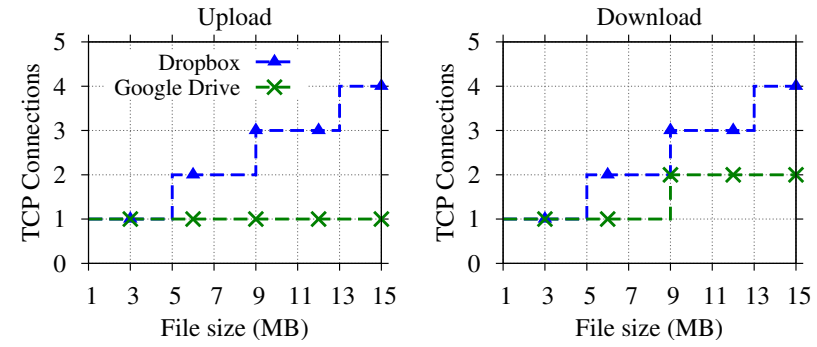
(a) Number of TCP connections



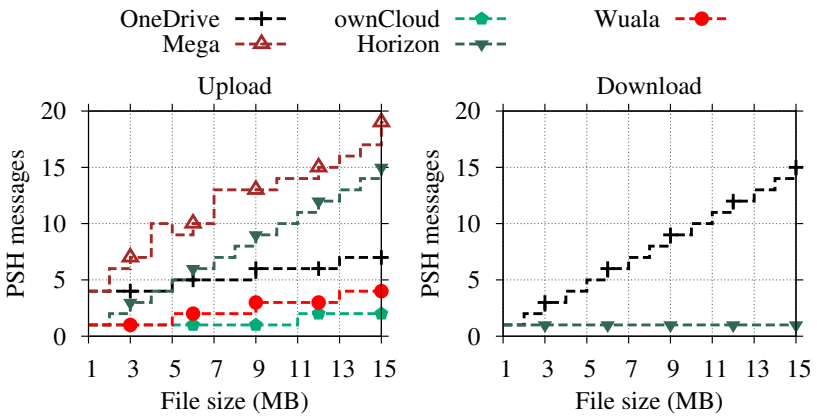
(b) Number of PSH-delimited messages

Figure 4.3: Bundling capability. Note the logarithm y-axes.

the application-under-test sets the PSH flag – hereafter called PSH-delimited messages. Notice how the number of PSH-delimited messages is proportional to the number of files in each file set. This suggests that each file is transferred as a separate entity, and no bundle is created. Timestamps of TCP PSH-delimited messages indicate that transactions are serialized in most cases, although some services (e.g., OneDrive) try to overcome this limitation by using concurrent transfers. Sec. 4.4 will show that services lacking bundling and performing transactions sequentially suffer performance penalties when the RTT is large and several small files have to be exchanged. Finally, among the 11 tested services, only Dropbox implements a file-bundling strategy [15].



(a) Number of TCP connections



(b) Number of PSH-delimited messages

Figure 4.4: Chunking capability.

4.2.2 Chunking

Large files can be either monolithically transmitted to the cloud or chunked into smaller pieces. Chunking is advantageous because it simplifies recovery in case of failures: Partial submission becomes easier to be implemented, benefiting both users connected to slow networks, and those having operations interrupted by temporary network disconnections. As in the previous experiment, by monitoring the number of TCP connections and PSH-delimited messages during the synchronization of files of known size, we determine whether chunks are created and transmitted in different transactions.

Our methodology reveals a variety of chunking strategies, some of which are illustrated in Fig. 4.4. Top plots (Fig. 4.4(a)) show that Dropbox and Google Drive implement chunking and rely on different TCP connections to submit individual chunks. In fact, Google Drive uses 8 MB chunks, while

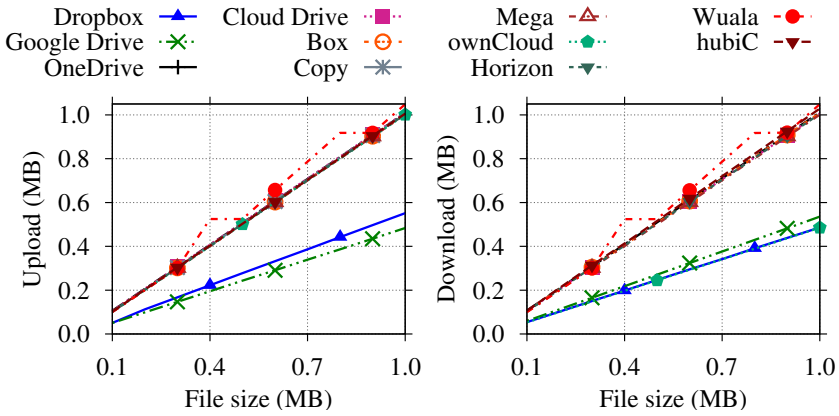


Figure 4.5: Bytes exchanged during the test with compressible files.

Dropbox uses 4 MB chunks – notice the changes in the number of TCP connections as file sizes exceed multiples of these quantities. Surprisingly, Google Drive relies on chunking only for downloading content.

A second group of services reuses TCP connections, but the number of PSH-delimited messages allows us to conclude that files are also split into chunks. Fig. 4.4(b) suggests that ownCloud chunks content at 10 MB boundaries when uploading, Copy uses 5 MB chunks, OneDrive and Wuala use 4 MB chunks, and Mega chunks at less than 1 MB. Looking at the Fig. 4.4(b) bottom-right plot (download), we observe that OneDrive and Horizon use different policies for each direction. Horizon, for instance, uses chunks smaller than 1 MB only when uploading, but no chunking when downloading. Finally, remaining services do not seem to implement chunking, since no difference is noticed when file sizes are varied (not shown in Fig. 4.4).

4.2.3 Compression

We next verify whether data is compressed before a transfer. Compression could, in general, reduce traffic and storage requirements at the expense of local processing time. We check the compression capability by contrasting the number of Bytes observed in the network with the original benchmark size when submitting highly compressible text files – sizes from 100 kB to 1 MB. Fig. 4.5 reveals that Dropbox and Google Drive compress data before transmission, with the latter implementing a more efficient scheme – i.e., less network traffic is measured when compared to the original file size. ownCloud compresses content only when downloading, whereas all remaining services send files as they are created.

Naturally, compression is advantageous only for some file types. It has a negligible or negative impact when already compressed files are going to be transmitted. A possible approach would be to verify the file format before trying to compress it – e.g., by checking file extensions or by looking for *magic numbers* in file headers. We check whether ownCloud, Google Drive and Dropbox implement smart policies by creating fake JPEG files – i.e., files with JPEG extension and JPEG headers, but actually filled with text that can be compressed. Results reveal that Google Drive and ownCloud identify JPEG content and avoid compression. Dropbox instead compresses all files independently of content and extensions. Hence, in the case of true JPEG files, resources are wasted.

4.2.4 Client-Side Deduplication

Server data deduplication eliminates replicas on the storage server. Client-side deduplication, instead, extends the benefits to clients and the network: In case a file is already present on servers, replicas in the client can be identified to save upload capacity. This can be accomplished by calculating a file digest using the file content (e.g., SHA256 is used by Dropbox [54]). The digest is sent to servers prior to submitting the complete file. Servers then check whether the digest is already stored in the system and skip the upload of repeated content.

To check whether client-side deduplication is implemented, we design the following experiment: (i) A file with random content is created in an arbitrary folder – since this is the first copy of the file, the full content must be transferred; (ii) the same random content is used to create a replica with a different name in a second folder – assuming hashes are used to identify replicas, only meta-data should be transferred, and not the complete file again; (iii) the original file is copied to a third folder – this step tests whether file names or any other information besides content hashes are checked to identify replicas in different folders; (iv) all copies are deleted and then the file is placed back to its original folder – this step determines whether deduplication still works after all copies of a file are deleted from local folders.

Results show three clear design choices. A first group of services (Box, Cloud Drive, Google Drive, hubiC, OneDrive and ownCloud) ignores deduplication opportunities and always submit the content even when files are available at both the client and the server side. In contrast, a second group of services (Copy, Dropbox, and Wuala) implements deduplication. Services in this group identify copies even after they are deleted and later restored. This avoids re-uploading the files in case they are restored in the local folder. In the case of Wuala, deduplication is compatible with local encryption – i.e.,

two identical files generate two identical encrypted versions. Finally, Mega and Horizon partially implement deduplication. Horizon keeps deleted files in a "trash bin" and restores files from it in step 4 of our experiment. Mega, in turn, identifies replicas only when they are added in a single batch.

It is also interesting to note that Dropbox used to implement inter-user deduplication. This technique allows a user to skip submitting files that are already stored by any other user. However, this scheme has been shown to leak information about content stored in the service [54]. By manually performing experiments with different users, we conclude that none of the services implement inter-user deduplication anymore.

4.2.5 Delta Encoding

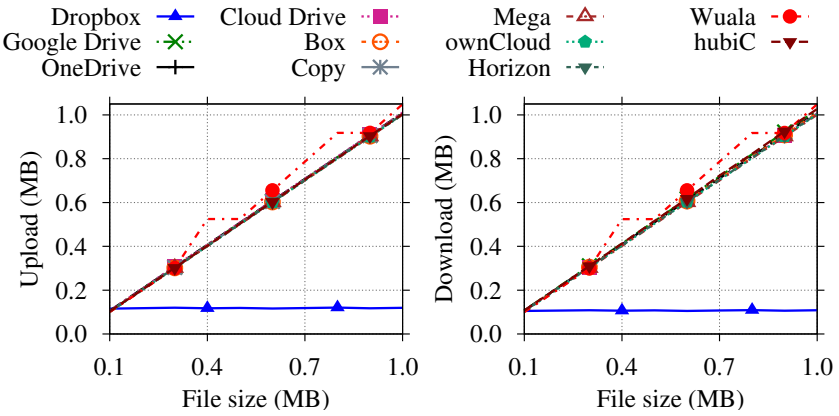
Delta encoding calculates the difference among file revisions, allowing the transmission of only the modified portions. Indeed, delta encoding provides similar benefits as the combination of chunking and deduplication, but with a finer granularity. It may have a positive impact on performance when files are frequently changed – e.g., when people perform collaborative work. On the other hand, the storage of static content is not affected by this feature.

To verify which services deploy delta encoding, a sequence of changes is generated on a file such that only a portion of content differs between iterations. Three cases are considered. New data is added (i) at the end; (ii) at the beginning; or (iii) at a random position within the file. This allows us to check whether any rolling hash mechanisms [71] are implemented. In all cases, the modified file replaces its old copy.

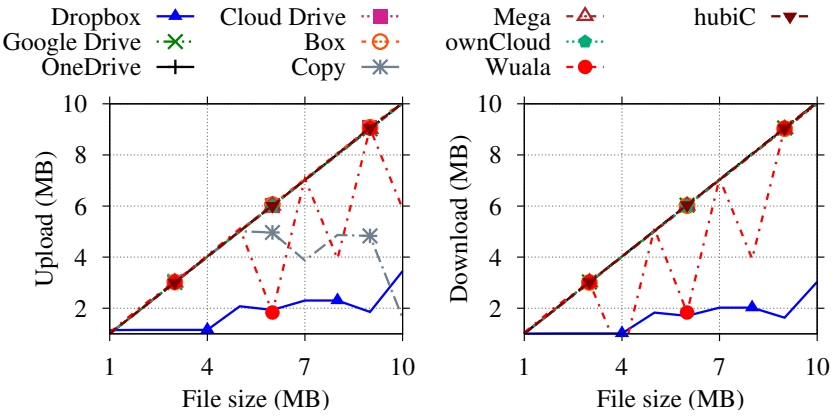
Fig. 4.6(a) shows the number of Bytes uploaded at each step of the experiment when data are appended to the file. File sizes have been chosen up to 1 MB, and 100 kB are appended at each iteration. We see that only Dropbox implements delta encoding – e.g., the volume of uploaded data in Fig. 4.6(a) corresponds to the actual part that has been modified.

Side effects of *chunking*, however, might increase the sent traffic in certain circumstances. Consider the results in Fig. 4.6(b). In this case, files of up to 10 MB are generated, and 1 MB of content is added at a random position within the file. Focusing on Dropbox again, observe that the amount of traffic increases when files are bigger than the Dropbox 4 MB-long chunk. This happens because the additional 1 MB can affect more than 1 chunk at once. For instance, adding 1 MB somewhere before the boundary of the first chunk would shift all data in the following chunks too. As such, the volume of data to be transmitted is larger than the new content.

Finally, we can see the trade-off of having either delta encoding or chunking/deduplication in Fig. 4.6(b). While only Dropbox implements delta en-



(a) Append content



(b) Insert content at a random position

Figure 4.6: Delta encoding tests. Note the differences in x-axes.

coding, chunking and deduplication allow Wuala and Copy to upload only those chunks that are actually affected by the addition of content at a random position. Notice how the number of Bytes exchanged by Wuala and Copy shows sudden drops in Fig. 4.6(b) when files are larger than the respective chunk sizes (4 MB and 5 MB). Yet, a major portion of files is uploaded again.

4.2.6 P2P Synchronization

Devices hosting common files could be synchronized without retrieving every content from the cloud, thus saving both network and server resources. Dropbox is known [15] for implementing a Local Area Network (LAN) Sync

Protocol that allows devices, possibly from different users, to exchange content using P2P communication when clients are connected to the same LAN.

To check which services implement P2P synchronization, we perform the following steps. We first manually verify each service Graphical User Interface (GUI) searching for explicit parameters related to P2P synchronization. This manual inspection reveals that both Copy and Dropbox offer such capability. We then configure these services to perform P2P synchronization and modify our testbed to put both test computers in the same LAN. We let the testing application submit workloads composed of single binary files of different sizes. By monitoring the traffic going to Test Computer 2 (downloads), we observe that only Dropbox and Copy indeed skip the download of content from the cloud, retrieving files from Test Computer 1. Other services, instead, always download the files from central servers, even if files are available locally at Test Computer 1, thus wasting Internet bandwidth.

4.2.7 Client Capabilities in Brief

Tab. 4.2 summarizes the capabilities of each service. It shows (i) whether bundling is implemented, either over multiple TCP connections or over multiple application layer transactions; (ii) the chunking strategy used to handle large files, including chunking thresholds; (iii) whether content is compressed always, never, only when downloading, or based on file formats (i.e., smart); (iv) whether the service implements deduplication, completely or partially; (v) whether the service implements delta encoding; and (vi) whether P2P synchronization is available. Dropbox is the most sophisticated service from the point of view of features to enhance synchronization speed. Other services have a mixed design, implementing a subset of capabilities. Finally, we see also very simple client designs in which no capabilities are present.

4.3 System Design

Next, we want to understand how services are implemented in terms of protocols and data center placement. In particular, we check the network fingerprint of each service when in idle state and whether services use a centralized or a distributed data center topology. Findings are summarized in Tab. 4.3.

Table 4.3: Summary of system design choices of each service.

Service	Topology	Data Center Location	Pooling Interval	Background Traffic
Mega	Partly distributed	Europe/Oceania	*N/A*	< 0.01 kb/s
Google Drive	Distributed	Worldwide	30 s	0.03 kb/s
Wuala	Centralized	Europe	5 min	0.05 kb/s
OneDrive	Partly distributed	U.S./Asia	45 s	0.05 kb/s
Dropbox	Centralized	U.S.	60 s	0.10 kb/s
Copy	Centralized	U.S.	60 s	0.12 kb/s
Box	Centralized	U.S.	60 s	0.16 kb/s
Cloud Drive	Partly distributed	U.S./Europe	5 min	0.72 kb/s
hubiC	Centralized	Europe	60 s	0.76 kb/s[2]
Horizon	*N/A*[1]	*N/A*	60 s	1.04 kb/s
ownCloud	*N/A*	*N/A*	30 s	1.24 kb/s

[1] *N/A*: Not applicable.

[2] hubiC figures vary considerably as more files are added – e.g., it reaches 12 kb/s when 87 MB distributed in 462 files are synchronized.

4.3.1 Protocols and Overhead

The traffic of the analyzed services can be roughly classified into two main groups: *control* and *data storage*. Regarding control, servers perform three major tasks: authentication, file meta-data control, and notification of file changes. As mentioned in Sec. 4.1, all services exchange traffic using HTTPS, with the exceptions of ownCloud and Dropbox notification protocols that rely on plain HTTP. Interestingly, some Wuala storage operations also use plain HTTP, since users' privacy has already been secured by local encryption.

We notice relevant differences among the services during login and idle phases. Fig. 4.7 reports the cumulative number of Bytes exchanged in a 40 min interval after starting the services and without synchronizing any files. For improving visualization, we report only services that exchange more than 90 kB on the interval. Two main considerations hold. Firstly, the services authenticate the user and check if any content has to be updated. Note how Cloud Drive exchanges about 300 kB with remote servers. Coming next, we can see that OneDrive requires about 150 kB in total, 4 times more than others. This is due to the service contacting many Microsoft servers during login. Other services produce a negligible amount of traffic at start up, thus reinforcing the weaknesses in Cloud Drive and OneDrive start up protocols.

Secondly, once login is complete, services may keep exchanging data with the cloud. The majority of services maintains an open TCP connection with servers, in which file changes are announced. In general, we observe silent protocols that seldom exchange any data in the network – i.e., changes are

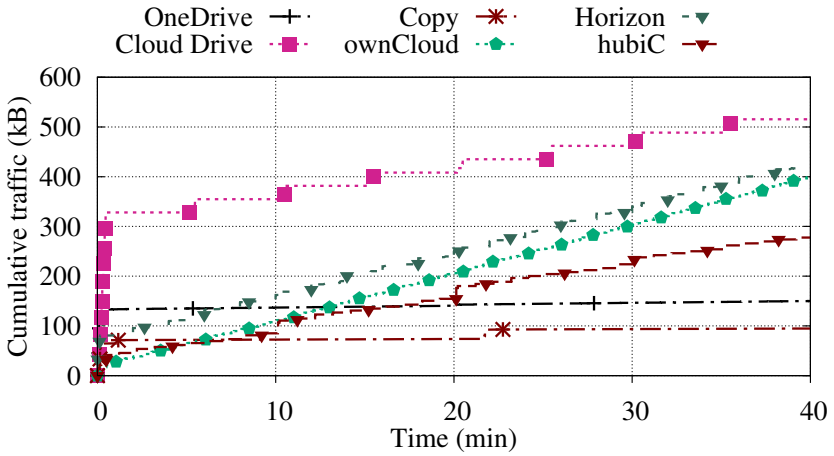


Figure 4.7: Background traffic generated by services in idle state.

notified by clients or servers as they occur via the open connection, and keep-alive messages are exchanged periodically only to guarantee that the notification channel remains open.

Copy, Dropbox and Box poll servers every 60 s, resulting in a bandwidth usage of less than 200 b/s. Similarly, Google Drive, OneDrive, and Wuala present negligible background traffic, but they pool servers at 30 s, 45 s and 5 min, respectively. Mega keeps a notification connection always open without exchanging any keep-alive messages at the application layer. It instead renews the notification connection after 10 min or more, thus putting almost no load in the network.

Other services, on the contrary, have noticeable traffic when idle. Fig. 4.7 shows that both Cloud Drive and hubiC generate more than 0.7 kb/s. Horizon and ownCloud are even less efficient: They produce 1.04 kb/s and 1.24 kb/s, respectively. Although apparently small, such background traffic might be problematic for users with limited traffic subscriptions and also for providers. For instance, 10 million users connected simultaneously to hubiC or Cloud Drive generate around 8 Gb/s even if no activity in the services is performed. Google Drive would consume less than 0.5 Gb/s.

Finally, we notice a clear problem in the notification protocol of hubiC. Its background traffic is proportional to the number of files in the service. For example, hubiC creates a constant stream of around 12 kb/s when 87 MB (462 files) are put in its folders during our experiments. These findings illustrate the relevance of designing notification protocols in cloud storage services.

4.3.2 Data Center Topology

A first group of services operates from a single region, relying on few data centers. Box and Copy are both centralized in the U.S., operating from San Jose and Detroit areas, respectively. Dropbox manages own servers also in the San Jose area. These servers are however in charge of control tasks only. All Dropbox users worldwide are directed to Amazon's data centers in Northern Virginia when storing data. Other two services are instead centralized in Europe, with hubiC operating from Northern France, and Wuala using two third-party locations, one in Northern France, and a second one in the Nuremberg area, Germany.

A second group of providers deploys a partly-distributed topology, in which few data centers are spread in key places in the world. Mega relies on third-party data centers in New Zealand and Germany. Cloud Drive uses Amazon's data centers in Ireland, Northern Virginia and Oregon. OneDrive takes advantage of Microsoft's data centers in the Seattle area (U.S. West Cost) and Southern Virginia (U.S. East Cost). We have identified IP addresses located in Ireland and Singapore as well, which seem to handle OneDrive control traffic, but no data storage. In general, traffic from our testbed is directed to the closest data center of each provider, as expected.

Finally, Google Drive is the only service with a fully distributed topology. Several edge nodes can be identified worldwide, and clients are redirected to the (possibly) nearest edge node based on DNS load-balancing. TCP connections are terminated at Google's edge node, and the traffic is then routed to actual data centers using Google's private network. We have identified more than 30 edge nodes spread in 6 continents. The advantages of such topology are twofold: (i) RTT between clients and servers is sensibly reduced, allowing for faster transfers over short-lived TCP flows; and (ii) storage traffic is offloaded from the public Internet.

4.3.3 System Design in Brief

Tab. 4.3 summarizes the design characteristics evaluated in this section. Services are sorted according to background traffic. We can see that Mega wins in this aspect: It implements the most efficient notification protocol and relies on data centers in different continents. On the other extreme, we see that hubiC and Cloud Drive produce noticeable background traffic, which might be a problem not only for users but also for servers if the number of on-line users becomes high. Finally, we observe that only major players besides Mega (i.e., Microsoft, Google, and Amazon) rely on global data center placement. Performance implications are discussed in Sec. 4.4.

4.4 Benchmarks

Table 4.4: Benchmarks to assess service performance.

Workload	Files	Binary	Text	Total Size	Replicas
1	1	1	–	100 kB	–
2	1	1	–	1 MB	–
3	1	1	–	20 MB	–
4	100	50	50	1 MB	–
5	365	194	171	87 MB	97 (5.4 MB)
6	312	172	140	77 MB	136 (5.5 MB)

We now evaluate the design choices by presenting a series of benchmarks. Sec. 4.4.1 describes the used workloads. Sec. 4.4.2 highlights the determining factors to the overall synchronization time, further detailed in Sec. 4.4.3, 4.4.4, and 4.4.5. Sec. 4.4.6 provides an overview of traffic overhead. Sec. 4.4.7 evaluates how the metrics vary in a long-term measurement campaign.

4.4.1 Workloads

The primary concern about workloads to be used for testing and performance assessment is to define them so that they are as close as possible to real-life scenarios, i.e., to design benchmarks that reproduce realistic workloads. To achieve this goal, we rely on the passive characterization of cloud storage usage presented in Chap. 3 and on previous works that investigate on Dropbox usage [15, 61, 65].

The main characteristics can be summarized as follows: (i) Files stored in the cloud are generally small, with typical data storage flows that carry few Bytes; (ii) the file size distribution is, however, heavy-tailed – users thus do generate some large content; (iii) the majority of content is already compressed (e.g., images, videos, and archives), although a non-negligible percentage (more than 30%) of compressible files is present (e.g., text files); (iv) files are often added in large batches with tens of files; (v) replication is common, both among folders of a single user and among different users because of shared folders.

Based on this information, we create 6 benchmarks varying (i) the number of files; (ii) file sizes; (iii) file formats; and (iv) the percentage of file replicas. Tab. 4.4 lists them. The first four sets are arbitrary and aim to test typical synchronization scenarios – i.e., small files, large files, bundles etc. The last two sets, instead, are formed by files recreated using a random subset of the

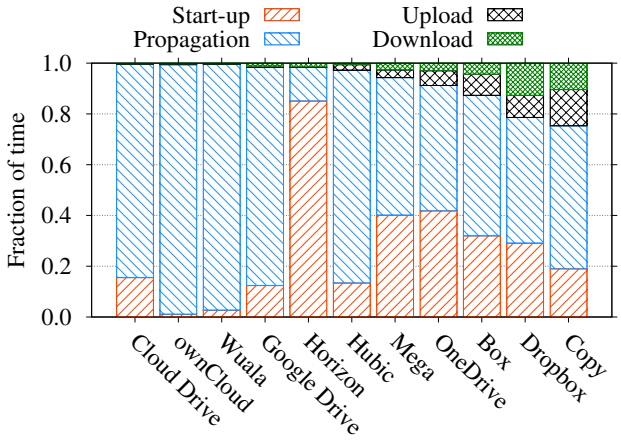
file metadata collected in [65]. They mimic a scenario in which a user adds a large number of files into a synchronized folder at once, such as when adding previously existing content or when migrating files between services.

All experiments are executed precisely in the same environment, from a single location, and under the same conditions, in order to isolate other effects and highlight the implications of design choices. Each experiment is repeated 20 times per service, and we wait until the network is completely silent to consider an experiment round complete. Furthermore, we run a different benchmark set (see Tab. 4.4) at each round and wait for a random period of time between rounds (always longer than 5 min) to prevent our results from being affected by systematic sampling bias [72] and to avoid creating abnormal workload to servers and the network. Only experiments that complete successfully are considered. Indeed, few tests have been aborted automatically due to failures, in particular with Wuala and Copy, for which reliability issues could be identified. Depending on the benchmark set and on the application-under-test, an experiment round lasts up to one day.

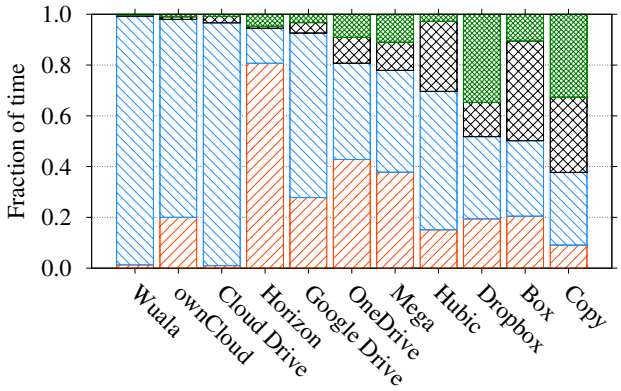
4.4.2 What Dominates Synchronization Delay?

We start by identifying which factors contribute the most to the overall synchronization delay. Figs. 4.8 and 4.9 report average values in each synchronization phase – i.e., the time interval in each step from the workload generation till the download is complete (see Fig. 4.2).

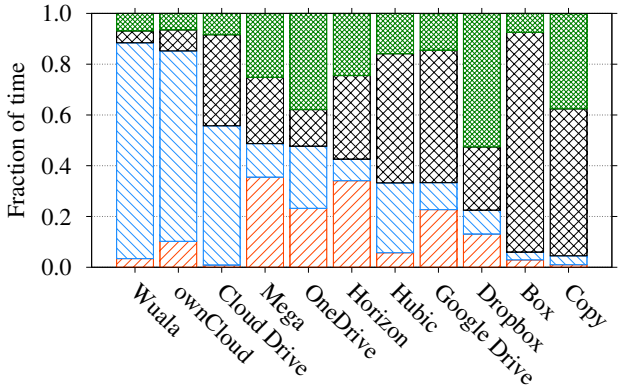
Fig. 4.8 depicts the fraction of time spent by the 11 services in each synchronization step for Workload 1 (1 file of 100 kB), Workload 2 (1 file of 1 MB) and Workload 3 (1 file of 20 MB), i.e., when a single small, medium or large file has to be synchronized. For improving visualization, we report only the part of the propagation delay between upload end and download start. Indeed, in this scenario, all services wait for the upload to complete before starting the download. Services are sorted by the sum of start-up and propagation delays – i.e., the most reactive services are on the right side.



(a) Workload 1 – 1 file of 100 kB

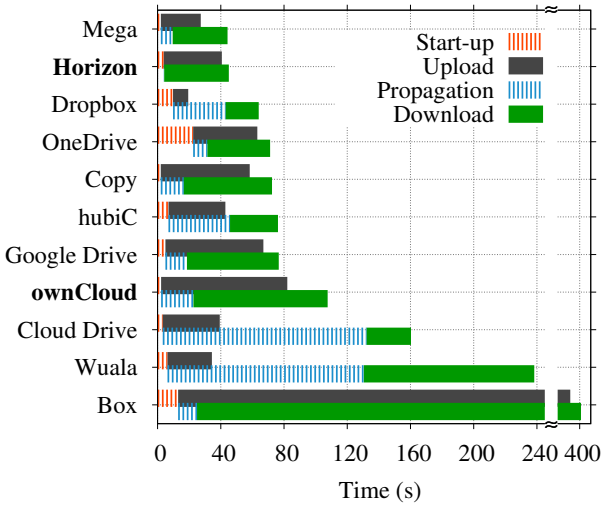


(b) Workload 2 – 1 file of 1 MB

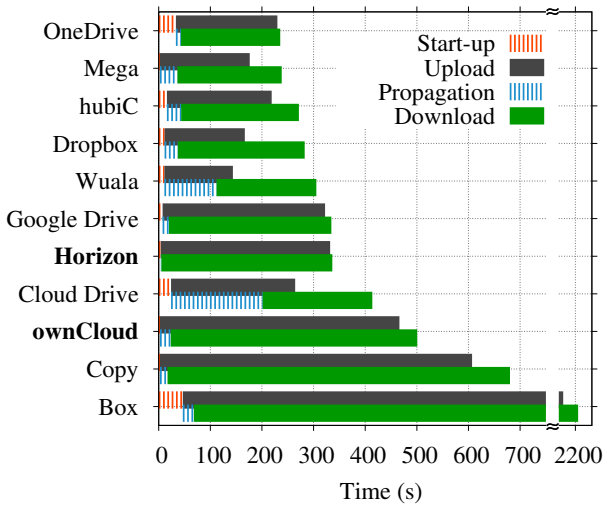


(c) Workload 3 – 1 file of 20 MB

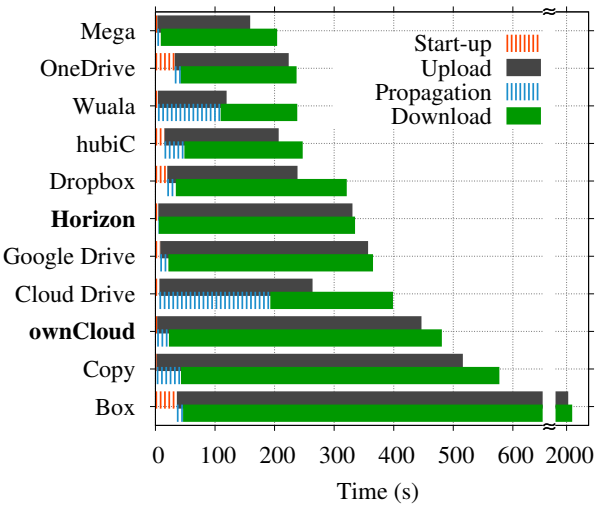
Figure 4.8: Fraction of time expended in each step of the synchronization cycle for single-file workloads. Note that clients are sorted according to their fraction of silent periods in each plot (i.e., start-up plus propagation delays).



(a) Workload 4 – 100 files of 10 kB



(b) Workload 5 – 365 files, 87 MB in total



(c) Workload 6 – 312 files, 77 MB in total

Figure 4.9: Average time to complete each phase of the benchmarks with multiple file workloads. Silent periods are marked with dashed bars. Services are sorted per total synchronization time. Note the discontinuity on x-axes.

A striking conclusion immediately emerges, in particular for near real-time scenarios (e.g., collaborative sharing of files). Despite the high variation among services, synchronization time in most examples is dominated by start up and propagation delays. This behavior might be expected for small files, where uploads and downloads complete very fast. Yet, for some services we observe that this holds true even with large files: Start up and propagation delays consume up to 90% of the total synchronization time when sending 20 MB files (see Wuala in Fig. 4.8(c)), whereas these delays are at least equal to 40% when sending 1 MB files in the best case (Fig. 4.8(b)).

As detailed later, some services exhibit clear performance penalties. Our local ownCloud installation, for example, takes longer to finish handling Workload 2 (1 MB file) than Dropbox – the former needs more than 10 s to complete a 1 MB synchronization, 98% of which is spent idle. Cloud Drive and Wuala require more than 2 min to complete the same 1 MB task. However, Wuala start up and upload delays are generally low, thus suggesting that the long delay is not due to Wuala local encryption. In both Cloud Drive and Wuala cases, the long delay is caused by a time-based approach to trigger downloads, which forces clients to stay idle for long intervals.

We repeat the analysis with multiple-file workloads in Fig. 4.9. Services are sorted according to their average total synchronization time. Overall, all services become slower when multiple files are involved. Mega and OneDrive win these benchmarks after expending 44 s, 235 s and 204 s on average to complete Workload 4 (100 files, 1 MB in total), Workload 5 (365 files, 87 MB in total) and Workload 6 (312 files, 77 MB in total), respectively. Interestingly, they generally achieve a good performance despite not implementing advanced client capabilities. In the case of Mega, this seems to happen because of both the proximity of its data centers to our test location and its lightweight client, whereas OneDrive performs well with large workloads because of its concurrent transfers. Dropbox with its sophisticated client is usually among the best as well as hubiC. Other services are much slower owing to several factors, including simplistic clients and traffic shaping policies as detailed later. Notice, for instance, that Box consistently requires one order of magnitude more time than others, expending 400 s to complete the synchronization of 100 files of 1 kB. Box figures grow to more than 35 min with Workload 5 and 6. Our private Horizon and ownCloud installations (marked in bold) present disappointing performance. Such poor performance definitively illustrates the importance of client design in cloud storage.

Overall, these results highlight that services have room for improvements, and better performance could be offered by improving system design. In the following, we present a detailed analysis of each synchronization phase to better highlight properties, strengths and weaknesses of each service.

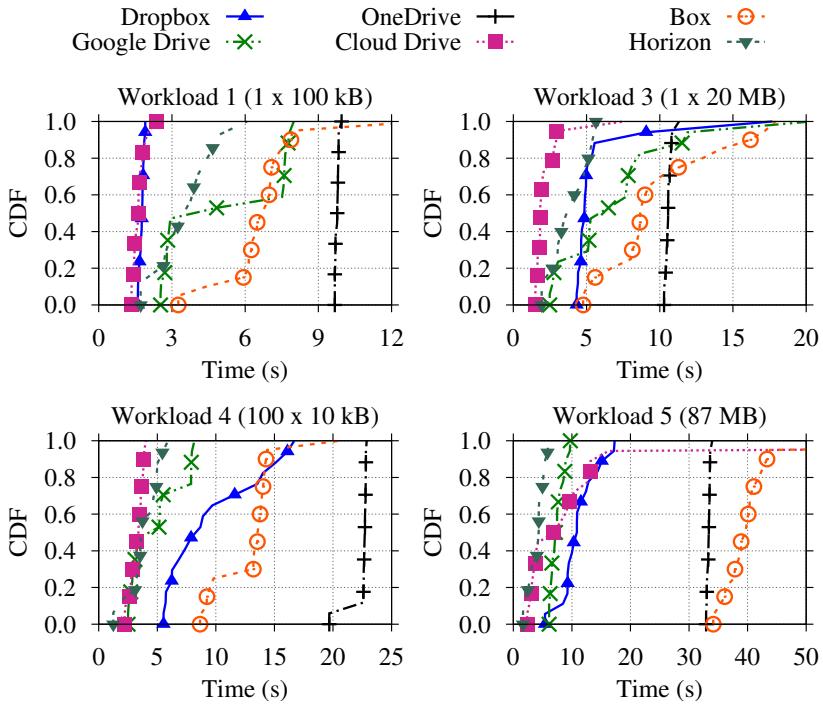


Figure 4.10: Start up delays. Note the different x-axes.

4.4.3 Synchronization Start Up

We investigate how much time services need before synchronization starts. This metric reveals whether the adoption of advanced capabilities (e.g., deduplication, bundling, etc.) could increase the initial synchronization delay. The metric is computed from the moment files are placed in Test Computer 1 until the first packet with payload in a storage flow is observed in the network – i.e., TCP and SSL handshakes are ignored.[1]

Fig. 4.10 depicts results for 6 services and 4 workloads. It shows the empirical CDF obtained by considering 20 repetitions. Other services and workloads are not shown for brevity, but the conclusions hold. When dealing with single small files (Workload 1, single file of 100 kB), we can see that Dropbox and Cloud Drive are the fastest services to start synchronizing. Dropbox's bundling strategy, however, delays the start up when multiple files are involved – compare to Workload 4, 100 files of 10 kB, in the bottom-left plot. As we have seen in the previous section, such strategy pays back in total upload time.

[1]The metric includes the delay of our application to send files to Test Computer 1. This artifact is ignored since all tests are equally affected.

Horizon needs few more seconds to start the synchronization even if servers are placed in our LAN. This large start up delay contributes to making the service slower than services operating from other continents, thus canceling advantages of the local installation. OneDrive is generally among the slowest, waiting at least 9 s before starting submitting files. The root cause of this delay is unclear since the service does not report activity during the period. Interestingly, Google Drive and Box CDFs present knees, also visible in other metrics. Coming sections will show that these services either have different servers that deliver variable performance or shape traffic explicitly.

Finally, start up delay is affected by the size of the workload being manipulated. We observe that the delay increases as either the number of files or the number of Bytes in the workload are increased. OneDrive and Box, for instance, take around 35 s and 40 s on average to start submitting Workload 5 (bottom-right plot). It is clear that performance could be improved by reducing the start up delays, in particular when large workloads are involved.

4.4.4 Upload and Download Duration

Next, we evaluate how long each service takes to complete upload and download tasks. This is measured as the difference between the first and the last packet with payload in storage flows going to Test Computer 1 (uploads) or Test Computer 2 (downloads). Again, we ignore TCP and SSL handshakes, tear-down delays and control messages sent after transfers are completed.[2] We expect the network properties to play a central role, i.e., services whose data centers are far from our test location are expected to show poor performance due to the well-known limitations of TCP with high RTT.

Focusing on top-left plot of Fig. 4.11, notice how services require a negligible amount of time to upload single 100 kB files. As the workload size is increased, upload time increases fast for services deployed far from our test location – e.g., Dropbox and Google Drive take up to 5 times more time than our local Horizon to complete 20 MB uploads (top-right plot). Box and Cloud Drive are even slower, the former requiring around 200 s to complete 20 MB uploads (800 kb/s, 170 ms RTT). When comparing results to download figures (Fig. 4.12), a different pattern emerges, confirming that Box and Cloud Drive shape upload rates only. Overall, we can conclude that data center placement and the explicit selection of system parameters are the determining factors for upload and download speeds with single-file workloads.

[2]Some services allow users to limit data rates manually. This functionality, as well as P2P synchronization, has been disabled where available.

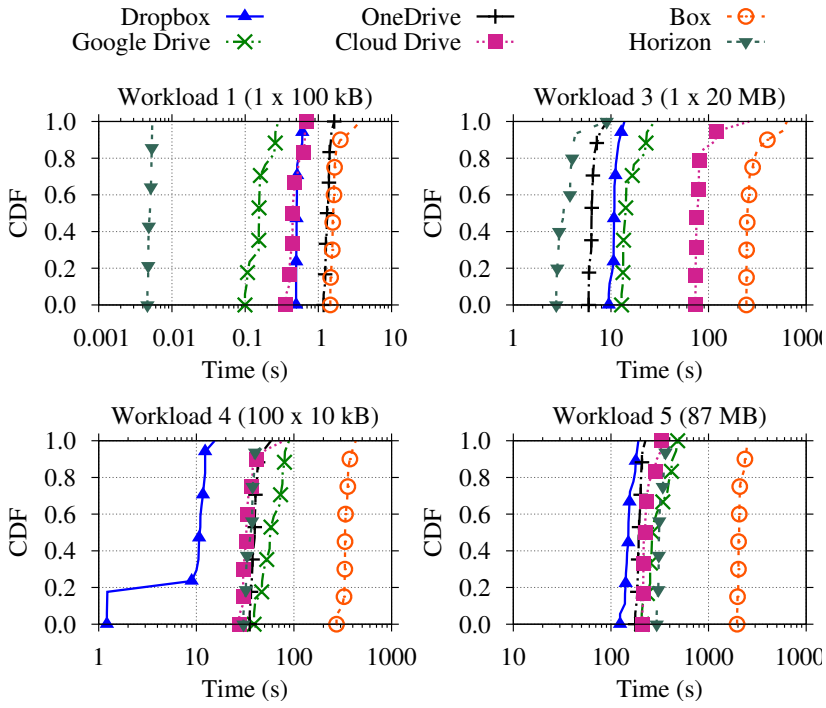


Figure 4.11: Upload times. Note the logarithmic x-axes.

When multiple files are stored, on the other hand, other aspects play the central role, such as client capabilities. Bottom plots in Fig. 4.11 and Fig. 4.12 show a striking difference on transfer duration when a large number of files is used. Notice how the upload and the download of 100 files of 10 kB (thus 1 MB in total) take more time than the exchange of a single 20 MB file in most cases. Dropbox wins when uploading Workload 4 (100 files, 1 MB in total) and Workload 5 (365 files, 87 MB) because its capabilities (e.g., bundling) allow the optimization the data transfer phases. However, Cloud Drive shows to be the fastest in the download direction for these two workloads. Dropbox remains at least two times faster than any competitor when sending Workload 4. We can thus see the trade-offs of implementing advanced client capabilities on Dropbox results: 20% of Dropbox uploads finishes in around 1 s. Those are, however, the experiment rounds in which Dropbox experiences its longest start up delays – see the tail of the distribution in bottom-left plot of Fig. 4.10.

Both Horizon (local installation) and Google Drive (world-wide data centers) are slower than Dropbox. They need around 300 s to complete experiments with Workload 5, whereas Dropbox finishes in 130 s, thanks to

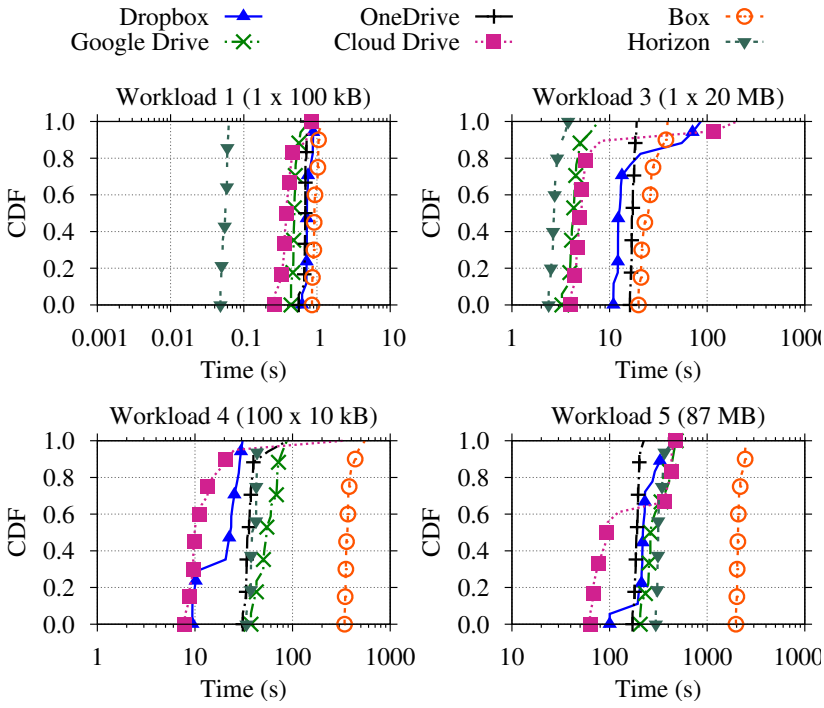


Figure 4.12: Download times. Note the logarithmic x-axes.

capabilities. OneDrive is the only service presenting a stable and consistent performance in both upload and download directions. Box shows again a poor performance when uploading because of the combination of traffic shaping with lack of capabilities. It takes more than 30 min to conclude experiments with Workload 5! Overall, results for other services reinforce the role of client design when manipulating complex workloads.

4.4.5 Propagation Delay

Propagation delays are depicted in Fig. 4.13. This metric is computed as the difference between download and upload starting times. It thus includes the first content upload delay, since none of the services is able to start downloading content before completing the upload of at least one file.

We see in Fig. 4.13 that propagation delays do not vary greatly among different workloads, with the exception of Workload 3, in which services wait for the single file of 20 MB to be uploaded before propagating changes. Google Drive, Dropbox, OneDrive, and Box are relatively stable and start downloading after 3–10 s in experiments with Workload 1, 4 and 5. Our Horizon

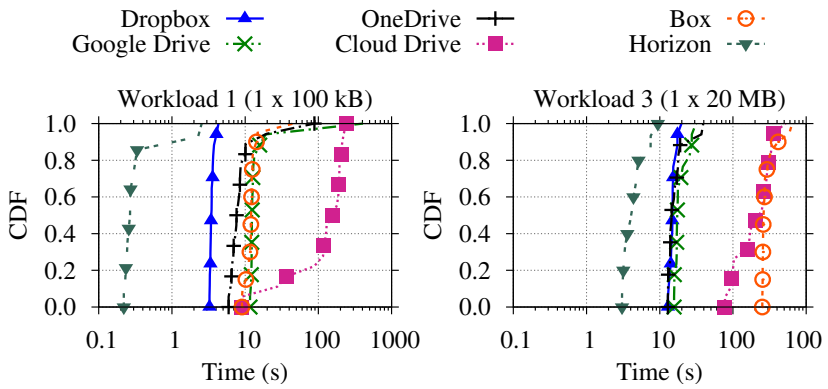


Figure 4.13: Propagation delays. Note the logarithmic x-axes.

installation is by far the most reactive. It notifies and triggers downloads almost instantly after upload is complete. Propagation delays of Cloud Drive vary greatly during the experiments. Manual inspection shows that Cloud Drive uses a time-based trigger of 5 min for checking for updates (thus waiting on average 2.5 min). In contrast, other services (e.g., Dropbox) implement asynchronous notification protocols. This behavior can be noticed by the shape of Cloud Drive CDFs in left plot of Fig. 4.13 – propagation delays grow from 10 till 300 s almost uniformly (note the logarithmic x-axes).

It is clear from these results that finding the right balance between start up, upload/download, and propagation delays is hard and our methodology helps in revealing trade-offs of the various design choices.

4.4.6 Traffic Overhead

We evaluate overhead to check to what extent the control traffic required for implementing client capabilities affects cloud storage network fingerprint. Fig. 4.14 shows the overhead CDFs of 6 services, calculated as the ratio between total exchanged traffic over twice the total file size (hereafter called *overhead ratio*). The factor 2 accounts for both upload and download phases. Note that the use of compression may lead to ratios smaller than 1 when text files are in the workloads.

We see that all services have a significant overhead when a single small file is synchronized (see left plot in Fig. 4.14). Services using one or several TCP connections for every file transfer, such as Cloud Drive, present the highest ratio. Dropbox also exhibits a high overhead, possibly owing to the signaling cost of implementing its advanced capabilities. On average, its ratio is equal to 1.2 when synchronizing 100 kB. Not shown in the figure, the overhead

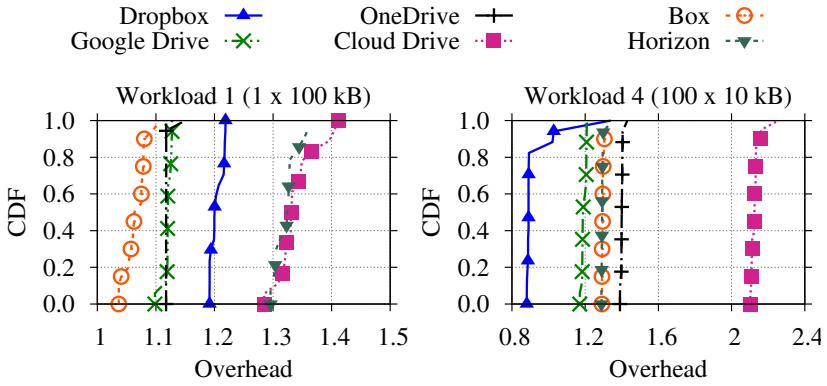


Figure 4.14: Traffic overhead. The x-axes depict the ratio between control and storage traffic over two times the benchmark size.

ratio decreases dramatically for most services when large files are submitted (e.g., Workload 3, single file of 20 MB). Dropbox surprisingly still presents high overhead (7%) in this scenario as well.

The lack of bundling dramatically increases the overhead when multiple small files are synchronized because the exchange of every file requires application layer control traffic to be sent. Cloud Drive, for instance, exchanges more than twice as much traffic as the actual data size when handling 100 files of 10 kB (Workload 4, see right plot). Services implementing compression naturally perform better with Workload 4, being 50% of its content compressible. The lack of bundling capabilities, however, makes the overhead of Google Drive to reach 20%, despite the compression gains. Google Drive indeed suffers from opening a separate TCP (and thus SSL) connection for each content. Compare Google Drive to Dropbox in Workload 4 to appreciate the extra cost. Other workloads confirm the advantages of client capabilities such as bundling, deduplication, and compression.

4.4.7 Long-term Delay Patterns

Finally, we perform long-term measurements to understand whether services present any performance periodicity. As we saw previously, services such as Google Drive deliver variable performance – e.g., sudden increases in upload times (see Fig. 4.10). We perform measurements with single file workloads over one month, committing more than 8,800 transactions. Fig. 4.15 shows results for uploads with 1 file of 20 MB. For improving visualization, we report the throughput in each transaction – i.e., benchmark size over upload delay. Results for 5 services are presented. Each experiment is marked with a dot

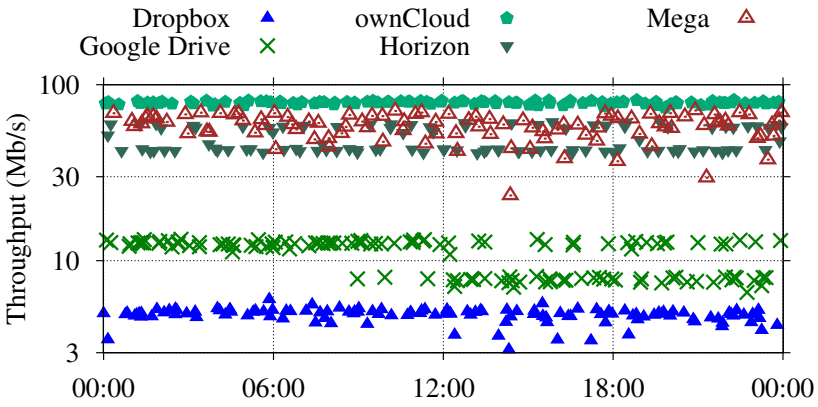


Figure 4.15: Upload throughput according to the time of the day (20 MB files). Note the logarithmic y-axes.

according to the time of the day it is performed. Thus, results are mapped to a single 24-hour x-axis.

We can see that services have stable performance without any evident periodicity. Google Drive is the only exception, exhibiting a bi-modal distribution: Experiments at night show 11 Mb/s of average upload throughout, while 8 Mb/s or 11 Mb/s is reached during day-time. Manual inspection reveals that synchronization is consistently slower when clients contact a subset of the Google edge nodes in charge of cloud storage traffic. Interestingly, slow nodes are likely contacted during daytime. Since the selection of edge nodes is automatically done by load balancing techniques, users have no control on what performance they will experience in a transaction.

Fig. 4.15 also reinforces the role of RTT when synchronizing large single-file workloads: Our local deployments (e.g., ownCloud) make a much better use of network capacity in such scenario than services operating far from our test location – e.g., Dropbox with 100 ms of RTT.

4.4.8 Benchmarks in Brief

Tab. 4.5 summarizes results. It shows: (i) An indication of client sophistication based on the number of implemented capabilities (Sec. 4.2); (ii) the average RTT from our testbed to servers, which is related to data center placement and topology (Sec. 4.3); (iii) the minimum and maximum traffic overhead ratio observed during the benchmarks, which is a consequence of the presence/lack of capabilities as well as the design of notification/control protocols; (iv) the time Test Computers spend idle before reacting in an

Table 4.5: Summary of the benchmark results.

Service	**Capabilities**	**RTT (ms)**	**Overhead ratio**	**Silent Period[1](s) and Total Synchronization Time[2](s)**							
				WL 2		WL 3		WL 4		WL 5	
OneDrive	+	129	1.00 – 1.40	21	23	28	46	32	71	43	**235**
Mega	++	45	**0.95 – 1.13**	**4**	**5**	**8**	**11**	10	**44**	36	**238**
hubiC	-	22	**1.01 – 1.12**	34	35	95	113	46	76	43	271
Dropbox	++++++	99	**0.93 – 1.09**	**7**	**10**	21	44	43	64	37	283
Wuala	++	35	1.00 – 1.89	135	136	147	158	131	238	112	305
Google Drive	++	12	1.00 – 1.19	18	19	27	32	19	77	20	334
Cloud Drive	-	46	1.01 – 2.13	165	167	232	254	132	160	202	414
Copy	+++	119	0.99 – 1.42	11	17	105	168	17	72	17	680
Box	-	170	1.02 – 1.30	29	33	321	347	25	406	68	2208
Horizon	++	< 1	1.00 – 1.30	**4**	**4**	**9**	**11**	**4**	**45**	**5**	336
ownCloud	+	< 1	1.00 – 1.22	11	11	22	24	23	108	23	501

experiment (i.e., *start-up* plus *propagation* delays), indicating how reactive services are; and (v) the total synchronization time for 4 different workloads. Services are sorted by the synchronization time in Workload 5. Bold highlights the best service in each metric.

While the RTT is a key parameter to performance as expected, the table reinforces the importance of design choices. We can see, for example, that although our private deployments win some benchmarks, other services deliver similar performance under much higher RTT, thanks to client capabilities, reactiveness, etc. It is also evident that there is no single winner. Sophisticated clients, such as Dropbox, seem to be the most suitable for near real-time scenarios (e.g., collaborative work), but the simple and concurrent design of OneDrive performs equally well with large workloads (e.g., migrations) despite the high RTT and reaction time.

4.5 Conclusions

This chapter presented a methodology to study personal cloud storage services through active experiments. It allows repeatable and customizable benchmarks to be executed without instrumenting proprietary services, i.e., adopting a black-box testing approach. The effectiveness of the methodology was shown in a case study in which 11 services were compared in an extensive measurement campaign. Our methodology unveils system designs, client capabilities, and assesses the time needed to synchronize devices. By contrasting the performance of local cloud installations and public services deployed in different continents, our case study showed the relevance of client capabilities, protocol design and data center placement.

We observed, for instance, that Dropbox outperforms our local ownCloud installation, even if the RTT from our testbed to Dropbox servers is three orders of magnitude higher, thanks to Dropbox sophisticated client. However, we also showed that client capabilities are not always a guarantee of good performance. Indeed, we concluded that the synchronization time of several services is dominated by silent periods, which could be reduced by engineering smarter notification protocols. Lightweight and reactive services, like Mega, were shown to perform better than other more capable services. Wuala and Cloud Drive, for instance, are strongly penalized by start-up and propagation delays in the order of minutes, due to notification protocol design.

Overall, our methodology and case study highlighted the implications of several design choices and the performance trade-offs engineers have to face when building cloud storage services. Our results are a strong indication that cloud storage can be improved, and better performance can be offered by fine tuning system designs according to the desired usage scenario. The tools implementing our methodology are available at [73].

Summary

In this part of the thesis, the focus was on cloud-based storage for personal usage. Internet users are attracted by such services due to their cheap (or even free) subscriptions and the ease they provide in synchronizing multiple devices, sharing files with friends, and backing up photos and documents.

We started our investigation on personal cloud storage with the passive analysis of three services of which we characterized the typical usage patterns presented by users. Results showed that a relevant amount of subscribers are accessing the service but exchanging hardly any data with it. On the other hand, few users exchange data volumes in the order of tens of GBytes per month. Interestingly, besides the client running on PCs, mobiles devices are being more and more used to access storage services as content producers, likely to backup photos and videos recorded on smartphones and tablets.

The characterization carried out through passive measurements allowed the understanding of typical usage in terms of exchanged data volume with the cloud back end. In the second half of our investigation, we leveraged such knowledge to define realistic workloads with which we assessed the performance of different storage services through automated and repeatable tests. For the purpose, we designed a testbed through which we identified the advanced capabilities implemented in proprietary clients (e.g., bundling, compression, etc.), and we measured the amount of time needed to synchronize devices under different workloads.

We noticed how performance is strictly dependent on the workload used, the distance (and consequent latency) towards storage data centers, and the implemented capabilities. All in all, a lightweight client reacting fast to changes is the key to good performance with small and medium workloads. Still, more advanced clients are essential when complex workloads are manipulated. The advantages of having a small distance to data centers are canceled if simplistic clients are used. In addition, we identified aggressive choices regarding the design of protocols to carry control notifications. Such choices lead to bandwidth wastage that might be significant for both users in limited capacity scenarios and busy servers.

Part II

Network Security Threats

Chapter 5

Network Security Threats

5.1 Introduction

Information security over the Internet remains a primary concern for consumers, enterprises, and governments alike. Malware infiltrates and spreads using complicated methods to hide its traffic among benign activities. Cyber-attackers continuously use sophisticated schemes to create new malware and evade detection by security measures, causing malicious activities to be on the rise on the Internet (nearly 200,000 new malware samples every day [74]). Recent industry reports disclose that zero-day vulnerabilities have increased by 61% [75], and antivirus software's detection rate of newly created viruses is less than 5% [18]. This is mostly due to the fact that existing security solutions, including Intrusion Detection Systems (IDSs), Intrusion Prevention Systems (IPSs), and antivirus tools, typically rely on honey-clients [76], content analysis [77] and blacklisting [78].

As a result, some malicious activities remain unnoticed for a long period of time, jeopardizing the security and privacy of users behind the compromised end points [18, 79]. Residential Internet subscribers are even more susceptible to such risks as they are not as protected as enterprise networks. Furthermore, the infected machines can be used by attackers for running malicious campaigns against other potential victims.

There has been an arm's race between security researchers and cyber attackers. Different approaches for malware detection have been taken by security practitioners coming both from the industry and the research communities. Such techniques range from instruction set and code analysis to traffic characterization of infected hosts. The result is a set of methodologies, each targeting either (i) a family of threats (e.g., Distributed Denial of Service (DDoS) attacks [80], botnets [81], phishing campaigns [82], click-

fraud [83, 84], Exploit Kit [85], or Drive-by Downloads [86]) or (ii) a specific phase of malware infection (e.g., exploitation, download, control). Often, the detector leverages specific features that, while effective for the targeted malicious activity, lose their effectiveness when targeting a different type of threat: Methodologies leveraging DNS traffic analysis are effective in detecting botnets that hide their infrastructure via fast fluxing[1] [87], but might not be as useful with Exploit Kits (EKs).

5.2 Detection Techniques for Malware Traffic

The literature suggests for a variety of techniques that can be employed in this context. We focus our attention on three macro-groups of detection techniques, being these the most related to our work. We discuss how previous works in the fields of (i) graph-based detection, (ii) multi-protocol traffic correlation, and (iii) infection phase identification relate to our research.

5.2.1 Graph-based Malware Detection

Previous work has explored graph-based approaches to detect malware. In [88], the authors build a bipartite graph consisting of domain names of failed DNS queries and host issuing such queries. Given this DNS failure graph, a graph decomposition algorithm is then applied to iteratively extract dense subgraphs. The intuition is that host infected by the same malware usually query for the same, similar or correlated set of domain names. The subgraphs generated are further classified in different categories and characterized by exploring their temporal properties. Similarly, building a relationship graph based on DNS historical data is proposed [89] where suspicious structural networks are identified based on two graph measures: Graph density and eigenvector centrality and ground truth labels. Recently, a malicious domain detection system [90] has been proposed. It leverages homophilic properties and ground truth labels to build a host-domain graph and adapt the belief propagation to estimate an unknown domain's likelihood of being malicious. Similarly, malicious hosts are detected using a semi-supervised, score-propagation algorithm that utilizes HTTP-communication graphs [91].

5.2.2 Multi-protocol Traffic Correlation

Many efforts have been focused on the analysis of a single protocol to identify network traces and patterns displayed by malware, while others have consid-

[1] The rapid change in the hostname of malicious servers to circumvent blacklisting.

ered a set of protocols to achieve the identification of malware. In the first case, HTTP and DNS are two of the most analyzed protocols among malware threats to communicate with victims or between malicious peers. Several detection techniques have been proposed, exploring different ways to characterize the behavior of different malware threats on HTTP [90, 92, 93, 94] or DNS [89, 95, 96]. Examples of proposals following a multi-protocol approach include [17, 97, 98, 99].

The popularity of HTTP on the Web has made it the preferred protocol for malware creators and as such, the target for researchers to detect and analyze malware. [92] presents a system to identify malicious drive-by download activities by exposing the distribution networks necessary to spread malware through HTTP. Similarly, [90] and [93] propose classifiers based on features from Web domains and URLs to detect malware activity. [94] proposes the use of n-gram techniques to filter out the majority of benign HTTP traffic and detect malicious HTTP transactions to be processed with more costly techniques. Systems that analyze the DNS protocol, usually look at failed DNS queries [89, 95], as this activity can lead to the existence of malware using Domain Generated Algorithms (DGAs). Other systems analyze the flow-level information from the DNS traffic and look for statistical patterns [96]. The problem with systems relying on a single protocol is their limited visibility (as malware can show activity over multiple protocols) and the required semantic understanding of the particular protocol being considered.

In comparison, other approaches have evaluated multiple protocols to detect malicious activity. A seminal work in this area is [97], where the life-cycle of botnets is modeled according to a set of phases, with different application protocols involved. An interesting approach is used in [17, 98, 99], where traffic information is presented through generic packet information such as length sequences and encoding differences, allowing then to represent the malware activity observed in different protocols.

5.2.3 Infection Phase Identification

Several types of malware usually exhibit specific phases during the infection process. For example, botnets are commonly used to distribute malware and involve several phases, including redirection of Web pages and communicating with a command and control server. Multiple approaches have looked for specific phases of the infection life-cycle, in order to detect the existence of malware. Examples include DNS queries failures [88, 89, 95], HTTP connections to domains [100, 101], and Command & Control (C&C) communications [96, 98, 102].

5.2.4 Limitations

Despite being of proven effectiveness to the context where they belong, all of the above techniques suffer from some limitations that could severely impair malware detection. Specifically, graph-based approaches focus on a sole protocol to identify the suspicious graph entities, detection strategies based on multiple protocols are tailored to a specific type of malware, and techniques based on infection phases are only able to detect malware at a specific point of its life-cycle. In other words, all these approaches restrict their effort to a specific condition to identify malicious activities, potentially losing visibility on other relevant evidence that would improve the detection capability.

In some cases, detection performance is assessed using synthetic datasets now outdated [103] or limited to specific types of attacks (e.g., ShellCode, Denial Of Service, CLET) [104, 105] that cannot be representative for the variety of threats affecting today's Internet. Moreover, the usage of honeyclients [76], i.e., hosts deliberately infected to analyze the traffic produced by malicious activities, can be the source of inconsistent detection techniques as sophisticated malware identifies such in-vitro condition, ultimately changing its behavior [106].

As malware nowadays constantly changes its behavior, relying on a single protocol, threat, or phase of the infection represents a strong limitation for detection systems, which calls for a different approach.

5.3 Proposed Solution

We aim at the ambitious goal of designing a flexible method that is agnostic about threat types, infection phases exhibited by malware, and protocols over which malicious communications take place. To do so, we follow an approach that is completely data-driven. We consider the actual traffic observed in a live network where users access the Internet. From the network flows, we extract and log *events* by means of passive monitoring tools. An event could be a HTTP request, a DNS response, or simply a TCP flow going to a remote host using an unknown protocol.

We consider a target event under analysis, that we call the *seed*, and we build a classifier that has to return a binary answer to the question: "Is the seed part of a benign or a malicious activity?" To provide the answer, we adopt data mining and machine learning techniques where correlation among events is leveraged to produce an augmented summary of the overall activity related to the presence of the seed.

We represent such summary in the form of a *Network Connectivity Graph*, CG in short, i.e., a graph where nodes and edges model the subset of events tightly correlated with the seed under study. The purpose of the CG is twofold. First, it provides a set of forensic information for the security analyst to support her in understanding the traffic involved in an accident. Second, using a supervised machine learning approach, the CG is used to extract a *model* of the typical behavior of malicious or benign events.

We name the proposed methodology MAGMA, MultilAyer Graphs for MAlware detection. MAGMA is a system able to process the huge amount of traffic coming from operational large-scale networks and to identify the subset of relevant events belonging to the same activity of the event under consideration, i.e., the seed.

Identifying such events is a challenging task, requiring ingenuity to correctly extract only the subset of them that are part of the seed activity while discarding the others. This is due to the fact that each host could be running multiple applications at the same time, each producing Internet traffic. Indeed, a user might be visiting a legitimate Web page, while malware running on her host is connecting to a C&C botnet. Additionally, the sequence at which events appear is typically non-deterministic, with randomness due to diversity (e.g., two hosts downloading the same page hosted on a CDN can show a different sequence of events in time or in space) and system state (e.g., a DNS request not appearing as the hostname has been previously resolved and cached).

MAGMA is based on a filtering and enrichment processes that leverages (i) temporal and (ii) spatial repetitiveness of events generated over time by multiple hosts. It looks for common patterns across different time snapshots produced by hosts connected to the network. This calls for a broader view of network activities that cannot be achieved using the point of view provided by single hosts. We, therefore, leverage a traffic trace collected from an operational network providing Internet connectivity to more than 20,000 subscribers. No traffic filtering is a-priori performed so to consider multiple network protocols and to face realistic cases of malware.

5.4 Contributions and Work Organization

The contributions of our work, MAGMA, can be summarized as follows:

- We characterize the traffic generated by malware in residential networks to testify the variety of malicious behaviors and the need for advanced tools to provide adequate countermeasure;

- we propose a methodology that extracts and represents the activity correlated with the occurrence of a *seed*, which allows the distinction between benign and malicious traffic;
- we provide augmented information to the security analyst to uncover hidden malware behavior supporting a thorough forensic analysis;
- we train a classifier that explicitly targets generic malware activities, and it is not tailored to a specific threat or malware class.

This part of the thesis is organized as follows:

Chap. 6 introduces the problem of malware in residential network by (i) describing the scenario faced during the data collection campaign (Sec. 6.1); (ii) characterizing the collected dataset and the volume of traffic produced by malicious activities (Sec. 6.2 and Sec. 6.3, respectively); (iii) identifying clear patterns in malware traffic (Sec. 6.4).

Chap. 7 describes our proposed solution MAGMA, MultilAyer Graphs for MAlware detection. (i) Sec. 7.1 introduces the proposed solution and gives a high-level view of the MAGMA system design together with the intuitions behind it; (ii) Sec. 7.2 describes the methodology designed to build the Connectivity Graphs detailing all the steps required; (iii) Sec. 7.3 provides a hands-on example of the evolution of the Connectivity Graph at each stage of the process, proving the capability of the methodology to produce a focused representation of relevant network activities; (iv) Sec. 7.4 shows multiple examples of Connectivity Graphs for various malware threats and contrasts them with the Connectivity Graphs obtained from legitimate traffic.

Chap. 8 builds on top of Connectivity Graphs to engineer a behavioral classifier whose performance is evaluated and discussed thoroughly. (i) Sec. 8.1 introduces the classification process; (ii) Sec. 8.2 details the motivations behind the choice of supervised classifiers and features considered for the purpose; (iii) Sec. 8.4 explains the achieved results and their dependence on the classifier type and on parameter setting during the Connectivity Graph construction process; additionally, (iv) App. A reports all the features considered for the classification process.

Chapter 6

A View of Malware in Residential Networks

In this chapter, we aim at characterizing malicious activities in residential networks. Hosts in this type of networks are usually equipped with traditional antivirus tools that are heavily signature or reputation based. Being installed on edge machines, they cannot achieve the macro-level view of the traffic in the core network and hence can not detect malicious behaviors in the wild or exploitation campaigns [92].

We instead aim at doing so by passively collecting network traffic at the PoP of an ISP. This approach guarantees two major advantages: First, we obtain a large-scale view of malicious activities, extending the knowledge available at single end-points or, worse, at honey-clients. Second, by analyzing traffic produced on a real and operational network, we face a huge variety of malware threatening users in home networks. The diversity of threat types and malware families observed is hardly reproducible in controlled scenarios.

In the following, we describe the scenario and the tools employed, we provide a thorough characterization of the recorded network traffic, and we present a large-scale measurement of malware in residential networks. We then investigate on malicious communications focusing on the traffic they produce and looking for behavioral patterns of different threat families.

6.1 Data Collection Scenario

We consider a vantage point located in a commercial ISP where approximately 20,000 customers are connected. Most of them are residential customers, connected via ADSL modems to the monitored point. Each customer's modem is given a static IP address, which is used to identify the

traffic generated/directed to all active terminals in the household. In the following, we use the term "user" to refer to traffic exchanged by a single household (IP address).[1]

A network traffic monitoring tool, namely Tstat (Sec. 1.2.1 – Network Traffic Analyzer), processes packets in real time, and produces a log file reporting TCP and UDP flows. For each flow, a record is stored detailing the flow identifier, $flow_{ID}$, the timestamp of the first packet, the total number of packets and bytes sent and received, the application protocol used, etc.

In case the application protocol is HTTP, the record further reports the server hostname, object path, user-agent, content-type, response status, and content-length. In case multiple HTTP transactions are present in the same TCP flow (e.g., due to HTTP-persistent option), multiple records are logged. Similarly, for each DNS transaction, the tool logs the requested hostname, the set of IP addresses returned by the resolver, or the response code in case of an error (e.g., *Non-Existent Domain*).

In parallel to the monitoring tool, a commercial IDS (Sec. 1.2.1 – Intrusion Detection Systems) processes the packets producing alerts if a network activity matches any rule in its database. For each alert, the IDS specifies the $flow_{ID}$ and a $threat_{ID}$, i.e., a numerical code that identifies a particular threat. For some $threat_{IDs}$, the name and the description of the malicious activity is available, detailing the severity of the threat and which component of the host is vulnerable, e.g., browser, OS, etc. Other $threat_{IDs}$ are instead little documented.

The IDS is very conservative in triggering alerts, and hence it is possible that some malicious events do not trigger any alerts (i.e., false negative – see Sec. 1.2.4). Conversely, every alert raised is related to malicious activities. In general, we cannot exclude that some few false positives are present. However, these appear to be marginal. Being conservative is a design choice common to many IDS products that prefer to limit the number of reported alerts than overload the analyst with too many warnings. Unfortunately, this also limits the visibility on the incidents that we observe. In the following, we use the IDS as oracle, i.e., network events are considered benign or malicious according to the IDS labels.

By matching the flow identifiers, alerts are linked to records, so that records can be flagged as *malicious*. We refer to a *flag* as a log record for which the IDS triggered an alert, and to a *flagged user* as a user exhibiting at least one flagged record. A *non-flagged user* is instead a user for which no alerts are risen.

[1]Given the popularity of NAT at home, the ADSL modem IP address identifies traffic exchanged by all devices accessing the Internet at each customer household.

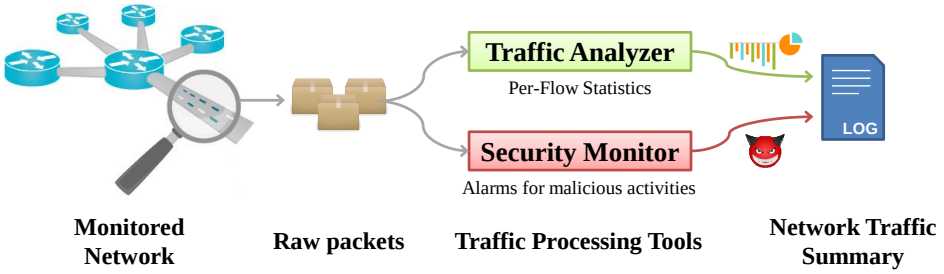


Figure 6.1: Network traffic capture workflow.

Table 6.1: Dataset summary for malware characterization.

Class	All Traffic		Flagged Traffic	
	Users (%)	Records (%)	Users	Records
HTTP	16,217 (*79.1*)	39.7 M (*11.8*)	1,308	42,007
Email	3,640 (*17.7*)	880.7 k (*0.2*)	-	-
Chat	3,045 (*14.8*)	100.8 k (*0.03*)	7	1,467
P2P	3,163 (*15.4*)	17.1 M (*5.05*)	-	-
OthTCP	18,806 (*91.8*)	22.7 M (*6.7*)	24	76
DNS	15,164 (*74.1*)	30.7 M (*9.3*)	-	-
VoIP	8,371 (*40.8*)	80.5 k (*0.02*)	-	-
OthUDP	17,664 (*86.2*)	224.6 M (*66.8*)	-	-
Total	20,486	336.1 M	1,321	43,550

P2P = (eMule, BitTorrent),
Email = (SMTP, POP3, IMAP),
Chat = (XMPP, YahooMsg, MSN, IRC)

Fig. 6.1 shows the overall process from raw traffic flowing through the monitored network to the log file produced by combining the records provided by the monitoring tool with the alerts raised by the IDS.

6.2 Dataset Description

For our analysis, we leverage a traffic trace obtained during one entire day in April 2012. We provide a detailed set of statistics to show the huge volume, heterogeneity and complexity of the data at our availability. Tab. 6.1 provides a summary of the dataset used for this study.

Focusing on the first three columns, we observe a total of 20,486 users generating about 336 million flows over the whole day. About 20% of those are related to HTTP and DNS records, with a large majority classified as

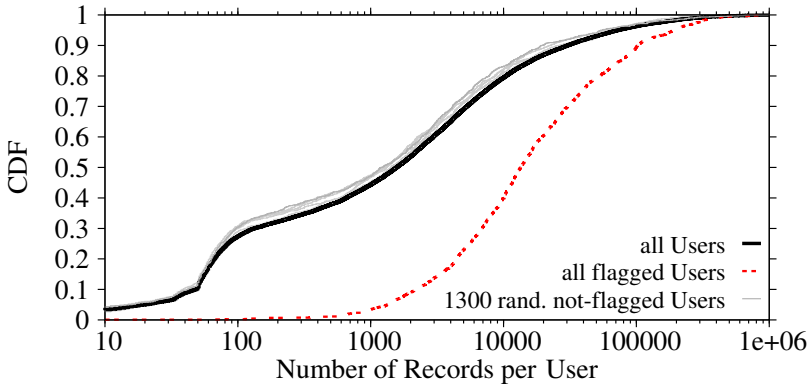


Figure 6.2: CDF of the total number of records per user.

"Other TCP" due to TLS/SSL (HTTPS) traffic, and "Other UDP" due to Peer-to-Peer applications. Rightmost columns, instead, detail the flagged records, i.e., records for which the IDS raised an alert.

Some traffic is machine generated, e.g., keep-alive messages, software updates, or cloud-based applications synchronizing their status. Some use proprietary protocols and generate little traffic. Other exchange information frequently inflating the number of events. Considering user-generated traffic, we observe some heavy users that generate thousands of HTTP requests, run P2P applications, play on-line games, and use multiple devices at the same time. Other users, instead, just have their mobile phone periodically checking the email.

Among all users, 1308 (6.4%) of them exhibit some malicious activity, with more than 150 different $threat_{IDs}$ being reported. Only 43,550 flags are raised by the IDS. That translates to a negligible 0.013% of all traffic. Most of these records correspond to HTTP traffic, with the exception of some Internet Relay Chat (IRC) and Remote Procedure Call (RPC) activities, which are known to be commonly abused by malicious adversaries. This highlights the very stealthy and low rate activity that malware is typically generating, and also confirms the conservative design of the IDS.

6.3 Network Traffic of Malicious Activities

6.3.1 Traffic Volume

We dig into more details to observe if it is possible to pinpoint differences between flagged users and non-flagged users. Specifically, we investigate on

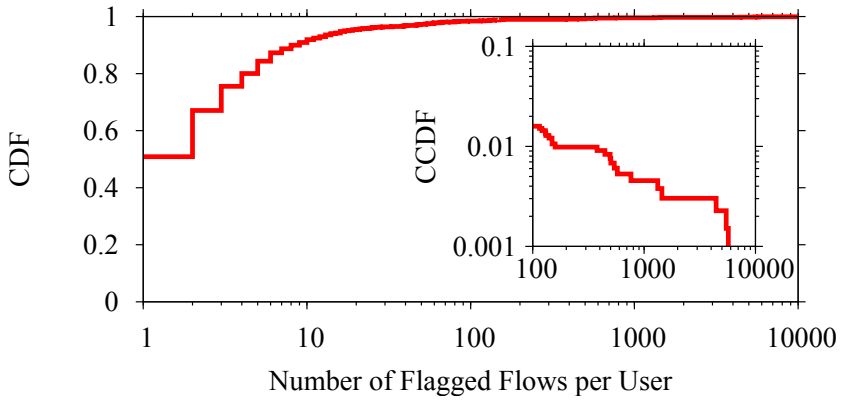


Figure 6.3: CDF of the number of flagged records per user.

the occurrences at which flagged events appear. The intuition is that the more flagged events occur, the easier it should be to spot them in the traffic aggregate. Fig. 6.2 shows the CDF of the amount of total records logged for all users, for the subset of flagged users, and for a random subset of 1,300 users among the not-flagged ones, so to be able to compare two populations of the same size. Results show that the flagged users generate much more traffic than the rest of the population. One would think this is due to the extra traffic generated by the malicious application running at the infected client. However, flagged users present a very small number of flags. This is detailed in Fig. 6.3 that reports the number of flags per flagged user. We find that 75% (92%) of the users show less than 3 (10) flags in the whole day, and only two users show more than 1,000 flags. As such, while malicious activities can inflate traffic volume, in general the rate of malicious records is very limited.

6.3.2 Threat Diversity

To investigate the threats diversity and the traffic they generate, Fig. 6.4 reports different statistics on the alarms raised by the IDS. Consider first the shaded histogram. It shows the number of users affected by each threat. Threats are sorted by popularity on the x-axis. Overall, the IDS detects 151 distinct threats. Their popularity is highly skewed, with the most popular affecting about 800 (61%) flagged users, and 129 threats affecting less than 10 users each. Despite the limited number of alerts, this highlights a very diversified scenario of malicious activities.

Next, consider the red dashed line of Fig. 6.4. It reports the CDF of the number of flagged records contributed by each threat. As expected, the ma-

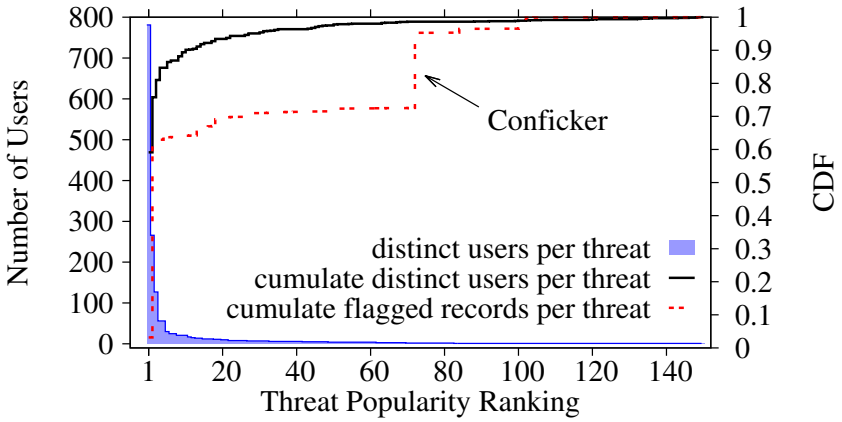


Figure 6.4: Overall threats statistics.

jority of these records are related to the most popular threats. However, the distribution has several steps, indicating that some are more "chatty" than others and produce many alarms even when only few users are involved. This is the case of Conficker [107], which infects only two users, yet it contributes to 23.3% of all flagged records. Note that Conficker is a worm that was first detected in 2008 but is still one of the most popular threats [108].

The solid line in Fig. 6.4 shows the CDF of the number of distinct users involved in each threat. In other words, we progressively add the fraction of new users that were not affected by previously considered threats. Notice how the distribution presents several "plateaus", indicating that all involved users were already accounted by previous threats. We find that 23% of users are flagged with multiple threats. This is due to users being infected by different malware.

To give more insights about how diverse and heterogeneous the malicious events are in the wild, Tab. 6.2 offers a deeper characterization of the 20 most popular threats. It details the popularity ranking of the threat, the number of infected users and the number of flagged records it generates. For some $threat_{IDs}$, the IDS provides limited information on the malicious activity and hence we adopt generic names, e.g., Threat-A. Notice how some threats presents a *type*. This corresponds to the ability of the IDS to identify different variant of the network traffic of the same threat.

Some examples of threats include Drive-by downloads and EKs, which are among the most popular threats. The *Dyndns activity* corresponds to traffic towards hostnames registered with Dyndns services that hide control messages (e.g., periodic communications to check network connectivity). *Skintrim* and *Tidserv* are two popular trojans that can trigger the download of other malware through backdoors. *Toolbar activity* threats are related to

Table 6.2: Top-20 most popular threats.

#	Name	Users	Flags	#	Name	Users	Flags
1	Drive-by download [type1]	781	1427	11	Threat-C	21	22
2	Dynactivity [type1]	266	26270	12	Toolbar activity [type2]	17	19
3	Blackhole EK [type1]	127	158	13	Drive-by download [type2]	15	33
4	Skintrim [type2]	56	301	14	Tidserv	14	228
5	Skintrim [type3]	56	301	15	Threat-D	14	470
6	Facebook plugin attack	30	31	16	unknown-I	12	263
7	Threat-A	25	27	17	iFrame injection [type1]	12	263
8	Blackhole EK [type2]	25	25	18	ZeroAccess EK	12	26
9	Toolbar activity [type1]	21	105	19	unknown-L	11	910
10	Threat-B	21	23	20	iFrame injection [type2]	11	260

the Ask.com toolbar that are triggered by the download of unwanted advertisement objects or perform `iframe` injections in the browser.

6.4 Patterns in Malware Traffic

For threats that have a large number of flagged users and records, we investigate whether there are "patterns", i.e., recurring events that would allow us to identify suspicious new threat activities. We focus on two examples as use-cases, namely Threat-D and Tidserv. Among all popular threats, these two present the largest number of flags per-user and a manageable popularity (about 10 users each). More importantly, they present different properties that highlight the complexity of the task we try to achieve.

6.4.1 Dissecting Threat-D

Let us consider Threat-D for which we do not have a priori knowledge of the malicious activities it relates to. Fig. 6.5 shows the whole day traffic evolution over time of the user having the highest number of Threat-D flagged records. Red triangles correspond to the events flagged as "Threat-D"; blue circles are events flagged as threats other than "Threat-D", while gray dots are "not-flagged" events. Each point corresponds to a different record whose client IP address is the same as the selected user.

We focus only on HTTP records by now. For visualization purposes, we consider an *HTTP event* defined as {server IP address, HTTP object name}. This causes two records, for instance `www.acme.com/main/index.html` and `www2.acme.com/secondary/index.html`, having the same server IP address (e.g., 10.0.0.1) to be mapped as the same HTTP event "`10.0.0.1/index.html`". The result is a compact representation of the traffic evolution over time where

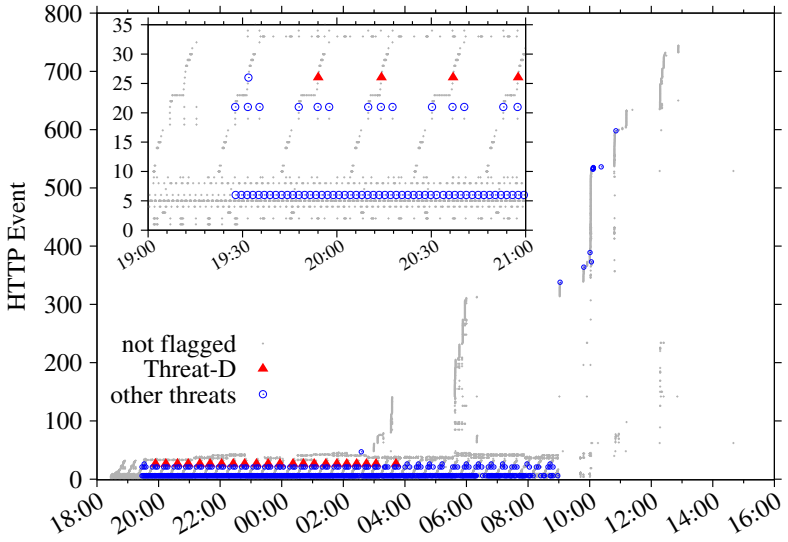


Figure 6.5: Traffic evolution of the most flagged IP address for Threat-D.

points on the same y-values indicate recurrent activities, and vertical shapes indicate different events happening in short amount of time (e.g., the user visiting a new Web page and fetching all its objects).

Notice the large periodic activity between 19:30 and 9:00. During this period, the IDS reports several alerts pinpointing specific events as shown in the figure inset. Tab. 6.3 further details all URLs requested in the first two hours (7.30–9.30 pm). While the user was (probably) not active (no vertical clusters of point are visible), almost all HTTP transactions relate to the same `instal/file.php` object hosted on different servers (all hostnames map to different IP addresses, thus generating different events). In other words, a sort of "Web scan" was happening over night. Further investigating the user-agent string, we correlated this activity to the Ask.com toolbar. We highlight that `instal/file.php` requests represent 78.2% of the overall user's requests. This shows that malicious activity can indeed inflate traffic volume as seen in Fig. 6.2.

Although the IDS effectively identifies the infected users, only a small subset of suspicious records are flagged. We verified that many suspicious HTTP requests failed (with a response code from the server "404 - Object not found"). These are not flagged by the IDS, indicating that the IDS rules consider both the client request and the server response to raise the alert.

Digging further among the events that are involved in this incident, we notice a lot of recurrent `www.google.com/webhp` requests. These are le-

Table 6.3: Requested URLs by the most flagged IP address of Threat-D.

URL	#
helpindownw.in/instal/file.php	689
notesonacocktailnapkin.com/wp-content/themes/rockstar/instal/file.php	99
www.usedtruckuk.com/wp-content/themes/classic/instal/file.php	93
www.google.com/webhp	73
www.creativelayer.net/modules/mod_wdbanners/instal/file.php	50
advantageclubrockford.com/modules/mod_wdbanners/instal/file.php	50
dreadneck.com/img/instal/file.php	49†
sotobetawi.com/wp-content/uploads/instal/file.php	46
www.veintisietelunas.com/images/stories/instal/file.php	40
www.usmctennis.fr/wp-content/themes/instal/file.php	40
www.radburnd.com/wp-content/themes/desk-mess-mirrored/instal/file.php	40
theuticashale.com/wp-content/themes/spectrum/instal/file.php	40
jiaotong.info/modules/instal/file.php	40
www.jlabmag.com/modules/mod_wdbanners/instal/file.php	25
www.google.com/	13
sas-rep.ru/space	10†
http://gv.t3.idspeed.com/tramp?ref=yonkis0	5
itjustdawnedonme.com/wp-content/uploads/instal/file.php	5
hannaoverstock.com/modules/mod_wdbanners/instal/file.php	5
cuteasabargain.com/wp-content/uploads/instal/file.php	5
bloggasaurus.com/wp-content/instal/file.php	5†
ashishpal.com/wp-content/uploads/instal/file.php	5
www.tpmedia.pl/wp-content/uploads/instal/file.php	4
www.tindellpictures.com/test_blog/wp-content/themes/instal/file.php	4
www.theuticashale.com/wp-content/themes/spectrum/instal/file.php	4
www.symzer.com/hey/wp-content/themes/instal/file.php	4
themarcellusshale.com/wp-content/themes/spectrum/instal/file.php	4
thegenieslamp.net/wp-content/themes/headlines/instal/file.php	4
datemit.com/instal/file.php	4
sasrep.ru/space	3†

† = flagged URLs

gitimate requests, possibly performed by the malware to run some simple network connectivity check.

Overall, this specific user presents a macroscopic evidence of temporal malicious patterns, considering both a strict definition (i.e., specific sequence of events) and a general one (i.e., repetitive behavior leading to a Web scan). The IDS provides strong hints of where to start to investigate, but some ingenuity and domain knowledge is needed to further elaborate the analysis.

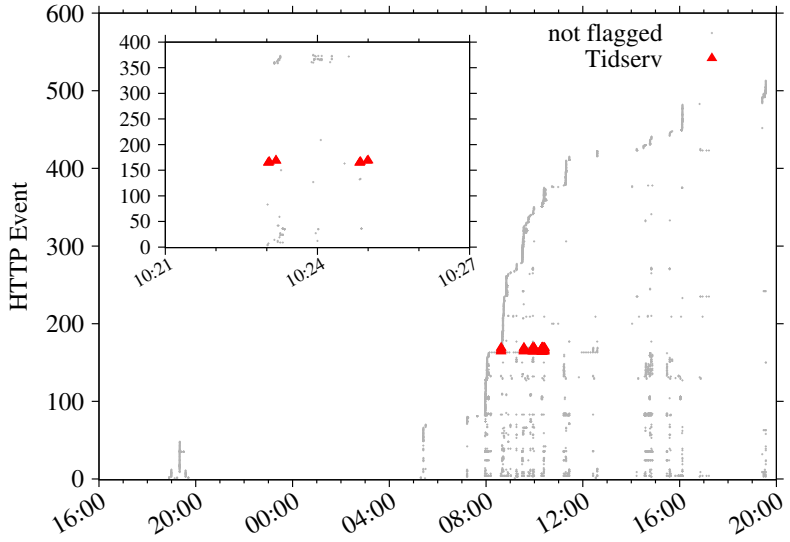


Figure 6.6: Traffic evolution of the most flagged IP address for Tidserv.

Unfortunately, we cannot generalize this result. In fact, other 13 users affected by Threat-D do not show the same traffic activity. Only another flagged-user presents one single `instal/file.php` object request. Other flagged records point to `ads.staticyonkis.com/www/delivery/afr.php` requests instead, which are happening in short time (up to 12 per second) with no trace of the Ask.com toolbar or connectivity check activities towards Google. In other words, Threat-D exhibits to different network behaviors.

6.4.2 Dissecting Tidserv

Consider the Tidserv flagged clients as another example. Fig. 6.6 shows the evolution of traffic of the user that has the highest number of Tidserv flags. A total of 59 flags are found, mostly happening in short time, as visible by the red triangles.

Focusing on the HTTP traffic produced and on the length of requested URLs plays a determinant role in the case of Tidserv. We find evidence of RapidDNS searches right before or after the alert raised by the IDS. Produced queries are related to advertisement download and possibly click fraud.

The length of flagged URLs is of primary relevance to detect the behavior of the thread under study. It is indeed possible to notice that all URLs are longer than 150 characters and present clear randomization and obfuscation components (e.g., *rlyg0-6nbcv.com/Kvb13* [...] *PWZhY2Vi*). Also in this

case, the IDS flags only a subset of the records that exhibit similar patterns, and URLs randomization is successfully hiding malicious events. Note that simply checking URL length or syntactic patterns of the random strings lead to both false negatives and false positives.

6.5 Conclusions

We provided a characterization of malware activities, starting from a broad view of their presence in residential networks and ending with discussing the technicalities of specific threats. Guided by a commercial IDS, we applied a bottom-up approach using the IDS alerts as clues indicating from where to start to look for patterns. The analysis was complemented by the information provided by the network monitoring tool, with which we highlighted the relevance of HTTP traffic properties, together with the need of a time reference to spot the correlation of network events over time. Analyzing the additional information available, we characterized the behavior of threats detected by the IDS and, more importantly, we identified activities that the IDS failed to detect. However, several issues and challenges remain.

Given the sophistication reached nowadays by malware, the manual investigation that we performed in the examples does not scale. We strongly believe in the need for methodologies to automate the analysis and augment the understanding of malicious activities found in networks, and especially in home networks where terminals are easily exposed to malware.

How to achieve this is an open research challenge. In the following chapter, we leverage as much as possible the network dynamics of malicious threats. Our goal is the creation of a system able to provide an enriched view of malicious evidences in network traffic. Starting from the hints provided by a reference IDS, the system extends the knowledge about the incident through the detection of other network activities related to the malicious event. We aim at the identification of high-level malicious symptoms, including atomic events (e.g., executable download), but also more complex relations among multiple events (e.g., a DNS scan triggering a click fraud) that are not covered by the specificity of IDS rules.

While a single one might not be sufficient to identify malicious activities, the combination of multiple symptoms together helps in making the picture clearer. Our system provides a comprehensive and automated methodology to leverage complex relationships and identify malicious patterns. As final result, it is able to classify a set of network activities as malicious or benign.

Chapter 7

MAGMA

MultilAyer Graphs for MAlware detection

In this chapter, we present MAGMA, MultilAyer Graphs for MAlware detection. It consists of a system that automatically extracts patterns of network activity related to a malicious event, i.e., a *seed*. Our system is based on a methodology that correlates network events of hosts normally connected to the Internet over (i) time (i.e., analyzing different samples of traffic from the same host), (ii) space (i.e., correlating patterns across different hosts), and (iii) network layers (e.g., HTTP, DNS etc.). The result is a *Network Connectivity Graph* that captures the overall "network behavior" of the seed. That is a focused and enriched representation of the malicious pattern infected hosts exhibit, purified from ordinary network activities and background traffic.

7.1 Overview

Our system has its roots in large-scale network traffic capture and analysis. For such reason, we consider a scenario in which a sniffer passively monitors the traffic generated by a large group of hosts, e.g., hosts in an enterprise network, or households connected to a PoP of an ISP. The sniffer extracts information from the packets and logs them in a file where each row corresponds to a different *event*. In parallel, an IDS tool analyzes the network traffic and raises alerts for activities that match any rule in its database, providing a $threat_{ID}$. The event logs produced by the sniffer are complemented with the information coming from the IDS, producing a final log of network events. Events can be either *malicious*, i.e., events for which the IDS raised an alert, or *benign*, i.e., no alerts from the IDS. Additional details about the traffic capture setup can be found in Sec. 6.1.

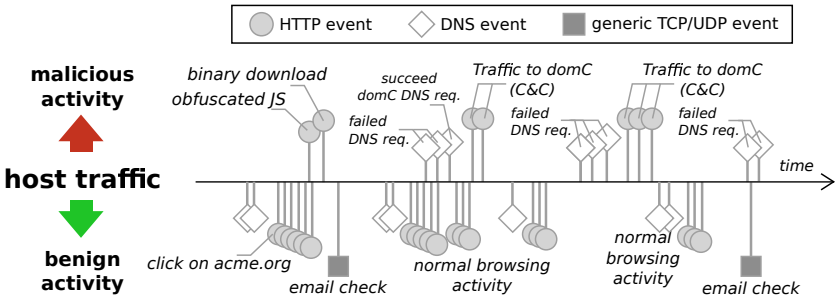


Figure 7.1: Example of events generated by a host as seen from the network.

7.1.1 Explicative Scenario

Consider the timeline of events generated by a single host reported in Fig. 7.1. DNS and HTTP events are reported using specific markers, while other protocols are reported as generic TCP/UDP events. The user is visiting some Web pages (e.g., `acme.org`), while an email client is polling a mail server for new messages. Benign events are reported in the bottom part of the timeline. Unfortunately, `acme.org` is hosting a Drive-by download page. Events on the upper part are due to the malicious activity in which the host is unknowingly fooled to download a malware from a malicious JavaScript contained in the Web page. We observe the download of the JavaScript, followed by the download of the malware executable. Once running on the host, the malware periodically contacts via HTTP a C&C server whose name is rotated among randomly-generated names [87]. The periodic polling is visible as a sequence of failed and successful DNS requests, and HTTP traffic to the C&C node.

Based on the view of the traffic from all monitored hosts, we design a methodology that extracts and characterizes common network activities. The challenge is how to isolate the events that are possibly correlated with a specific malicious activity from the "background" noise caused by other events. All the events are indeed exposed by the system, and, as we will see in what follows, only a handful of them are actually malicious.

7.1.2 Network Connectivity Graph

Consider a seed and the timeline around it. Intuitively, close-in-time events are likely to be related to it. For instance, in Fig. 7.1, the DNS request followed by several HTTP requests to the `acme.org` server could be identified as a typical pattern. However, Drive-by Download attacks [86] can mimic or be hidden in the same behavior. To isolate them, we study snapshots of traffic that contain the specific seeds.

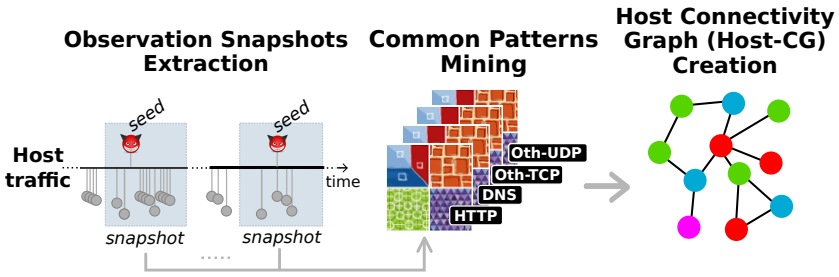


Figure 7.2: Host Connectivity Graph generation.

Fig. 7.2 shows the workflow used to transform the events of a given host into a *Host Connectivity Graph (Host-CG)*. Three steps are executed: (i) Snapshots extraction; (ii) per-layer common patterns mining; and (iii) Host-CG creation.

Snapshots Extraction.
For each instance of the seed, we extract a *snapshot* defined as the *ordered* set of events occurring in the temporal window centered at the seed. Two snapshots are presented in Fig. 7.2 as example.

Common Patterns Mining.
We look for commonalities across snapshots. In particular, we look for *patterns*, defined as the *unordered* set of events, that appear across multiple snapshots. Intuitively, the periodic HTTP requests towards the C&C server would possibly be a repeating pattern on the HTTP-layer. On the contrary, the Web browsing events asynchronously generated by the host would be present only in a small subset of snapshots.

We extract separate common patterns by processing the host traffic considering *layers* in isolation. The traffic generated on each layer corresponds to all events of a specific protocol so that HTTP, DNS, other-TCP (i.e., all TCP communication except HTTP on port 80), and other-UDP (i.e., all UDP communication except DNS on port 53) events are separately analyzed. This choice originates from the fact that each protocol has some peculiarities that we would leverage. For instance, in the HTTP layer, we are looking for common and repetitive patterns. On the DNS layer instead, a single failed DNS request may be more interesting than successful DNS requests.

Host Connectivity Graph.
For each layer, we represent the common pattern as a graph where nodes and edges are specifically defined to offer a compact yet rich representa-

tion. Consider the HTTP-layer. URLs can be represented by separating server hostnames and object paths using two nodes: An edge between the hostname and the path would thus represent a URL. The resulting graph captures the website structure. For example, `acme.org/index.html` and `acme.org/logo.png` are represented with a hostname node (`acme.org`) and two objects nodes (`index.html` and `logo.png`, respectively). Similarly, in the DNS layer, the request for `acme.org` is linked to the IP address(es) returned by the resolver.

As last step, we connect each per-layer graph into a final Host-CG. This is done by adding links across multiple layers. For instance, the `acme.org` hostname in the HTTP layer is linked to the same node in the DNS layer.

The resulting graph is a rich and compact representation of the common network activity related to a specific seed and a given host. Each layer brings a specific characterization of the activity given by a protocol, resulting in an overall integration of common patterns. In the previous example, the HTTP layer highlights the websites hosting the binary download, while the DNS layer reveals failing requests triggered before or after C&C communications. Focusing on each activity in isolation would miss such relationship.

Seed Connectivity Graph.
We leverage the fact that the same seed may be present in the timeline of different hosts, offering some "spatial" diversity. To have a broader view of the common activity related to a specific seed, we "fuse" multiple Host-CGs into a single *Seed Connectivity Graph (Seed-CG)*. As for the common pattern, we have the freedom to choose between a selective fusion, e.g., retaining only those common nodes from all Host-CGs, or a permissive fusion, e.g., merging all nodes from all Host-CGs.

7.2 Building the Connectivity Graph

This section discusses the design choices taken and the parameters to control when creating a Network Connectivity Graph. The pseudo-code in Alg. 1 details the overall approach.

7.2.1 Snapshots Extraction

The first step to process the traffic generated by each host (h) among the set of hosts (H) exhibiting the seed (s) is to extract an observation snapshot for each occurrence of the seed. We define the parameter Δ that controls the duration of the snapshots. In particular, a snapshot is composed of all events

Algorithm 1 Create Network Connectivity Graph.

input args	s: seed
	H: set of hosts
	Δ: snapshot duration
output	Seed Connectivity Graph

1: **procedure** graphLayer (s, S, layer):
2: P = *findCommonPattern* (s, S, layer)
3: **return** *fromPatternsToGraph* (P, layer)

4: **procedure** hostConnectivityGraph (s, h, Δ):
5: S = *getSnapshots* (s, h, Δ)
6: g_{HTTP} = *graphLayer* (s, S, 'HTTP')
7: g_{DNS} = *graphLayer* (s, S, 'DNS')
8: g_{TCP} = *graphLayer* (s, S, 'TCP')
9: g_{UDP} = *graphLayer* (s, S, 'UDP')
10: **return** *connectLayers* (g_{HTTP}, g_{DNS}, g_{TCP}, g_{UDP})

11: **procedure** seedConnectivityGraph (s, H, Δ):
12: $G_s = \emptyset$
13: **foreach** h $\in$ H:
14: $G_s \leftarrow$ hostConnectivityGraph (s, h, Δ)
15: **return** *fuseGraphs* (G_s)

occurring in the interval $\pm\Delta/2$ centered around the seed. In case consecutive snapshots overlap, we apply two strategies depicted in Fig. 7.3 to solve the conflict. If the overlapping window lasts for less than $\Delta/2$, the two snapshots are merged. Otherwise, the overlap is split into two halves, each associated to a different snapshot. These operations are executed by *getSnapshots*() (Alg. 1 line:5) that receives the seed (s), a host (h) presenting at least one instance of s, and the snapshot duration (Δ) as inputs. It returns the set of snapshots found (S).

Different values of Δ can lead to different results. In particular, the larger the Δ (e.g., hours), the more the snapshots will merge. This results in fewer snapshots on which perform pattern mining, producing "noisy" data since not many events are filtered. Conversely, a small value of Δ (e.g., seconds) might be too conservative. In the following, we set Δ = 30 minutes. A complete sensitivity analysis is reported in Sec. 7.3.4.

7.2.2 Patterns Mining

We use the frequent itemset mining technique to extract common patters [109]. This technique works on unordered sets of simple objects (e.g., strings).

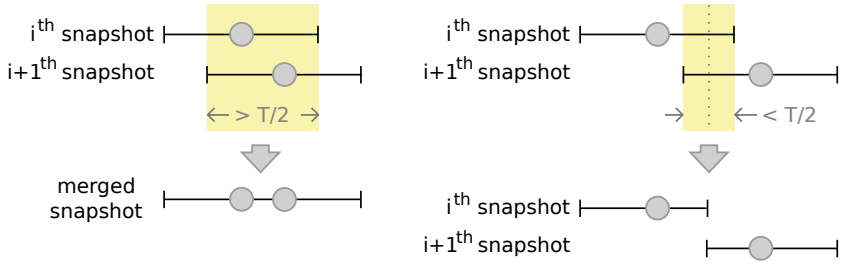


Figure 7.3: Snapshots creation policies when consecutive snapshots overlap.

Snapshots, however, correspond to ordered sequences of events that may appear multiple times. We thus map each event to an *item* based on the event properties. Specifically,

- a HTTP item is represented by HTTP URLs with parameters removed, e.g., `http://domain.com/path/file.ext`;
- a DNS item combines the requested hostname, and either the list of returned IP addresses or the query response error code, e.g., *DoesNotExists.com – NXDomain*;
- TCP and UDP items are represented by the server IP address and the server port being contacted, e.g., *10.20.30.40:443*.

For each snapshot, we create a *transaction* containing the set of *distinct* items. We look for common *itemsets*, i.e., sets of items common across multiple transactions. A *support* value is computed for each itemset and indicates the fraction of transactions containing the specific itemset. For a given support value, the itemset presenting the highest number of items is said to be *closed*. The closed attribute implies that there is no other itemset made by more items with the same support. An itemset is "frequent" if its support is greater than or equal to MinSup.

Itemsets with a number of items smaller than MinLen could be discarded. By setting MinLen=1, frequent itemsets are equivalent to simple frequent items. For MinLen=2, at least pairs of items are considered. For instance, consider `acme.org/index.html` and `acme.org/logo.png` that appear in 70% and 45% of snapshots, respectively. The itemset (`acme.org/index.html`, `acme.org/logo.png`) may appear from 15% to 45% of snapshots.

Looking for all itemsets is a #P-hard problem [110], but well-known algorithms efficiently compute frequent closed itemsets. We rely on the Carpenter algorithm [111], which is specifically designed for datasets made of few transactions (i.e., snapshots) that have a huge number of items (i.e., events).

MAGMA looks for *frequent closed itemsets* that, for simplicity, we call *patterns*. Patterns are extracted by *findCommonPatterns*() (Alg. 1 line:2), that receives the seed (s), the set of snapshots (S) and the layer (layer) to process. It returns the pattern (P). The pattern extraction process is guided by the definition of the value of MinSup: All events that do not appear with frequency at least MinSup are discarded. We set MinSup = 1/2, i.e., for each host, we discard all events not appearing in at least half of the snapshots. Sensitivity analysis in detailed in Sec. 7.3.4.

7.2.3 Host Connectivity Graph

As previously discussed, we individually process each layer to create separate graphs. The *graphLayer*() (Alg. 1 line:1) extracts patterns for a specific layer and maps them into a graph. This mapping exploits a subset of the events properties, as follows:

- The HTTP layer has two types of nodes, hostnames and object paths. An edge connects the hostname and the object path to compose a URL;
- the DNS layer has three types of nodes: Server hostnames, server IP addresses, and DNS error codes. An edge connects the hostname to either the IP addresses returned by a DNS response, or to an error code;
- the TCP and UDP layers have two types of nodes: Server IP addresses and server ports. An edge connects the two to represent a TCP or UDP connection.

Different graph layers are combined in a single Host-CG using *hostConnectivityGraph*() (Alg. 1 line:4). The function starts by extracting the snapshots (S) related to the seed. The shapshots are then processed to extract the graph layers (g_{HTTP}, g_{DNS}, g_{TCP}, g_{UDP}) through calls to *graphLayer*(). The separate graph layers are finally integrated to form the Host-CG using the *collectLayers*() function, which looks for common nodes across the layers and links them as represented in Fig. 7.4. Notice that each graph layers contains the host (h) IP address by construction.

Using this procedure, the HTTP layer is augmented with the set of server IP addresses being returned by the DNS layer. In this way, for example, the HTTP layer can acquire few of the network/service architecture (e.g., many IP addresses associated to a hostname are typically indication that the contacted service is hosted on a CDN). Similarly, the contacted IP addresses on the TCP and UDP layers are linked to hostname (when available).

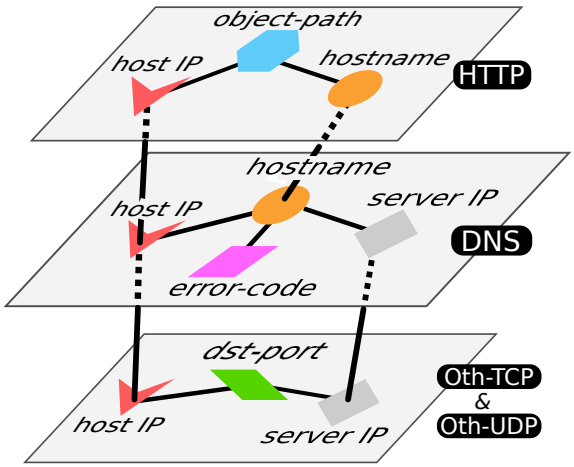


Figure 7.4: Graph layers nodes and multi-layer connections.

7.2.4 Seed Connectivity Graph

To provide the global view of the common behavior gained by observing multiple hosts, we combine all Host-CG. This operation is performed by the *seedConnectivityGraph*() (Alg. 1 line:11) function. For each host (h) among the subset presenting the seed (H), the function creates the Host-CG calling *hostConnectivityGraph*(). All the output graphs are collected into the set G_{hosts}. The graphs are finally merged using *fuseGraphs*().

This operation can consider different strategies. For instance, applying a strict intersection would retain only nodes appearing in all Host-CG. In the worst case, this results in a Seed-CG containing only the original seed. More complex strategies can instead compute node and link frequency or popularity among hosts, and discard those below a given threshold of MinPopularity.

In the following, we consider the strict intersection across Host-CG as the default choice, i.e., MinPopularity=1. A detailed discussion about the impact of this choice is deferred to Sec. 7.3.4.

7.3 Connectivity Graph Characterization

In this section, we identify the network events eligible as seeds for MAGMA and we built the Seed-CGs for them. We also show how CGs evolve along the different steps of the methodology described in Sec. 7.2. Finally, we evaluate the benefits and properties of CGs created by MAGMA showing their representativeness using both malicious and benign seeds.

Table 7.1: Eligible seeds with minSnapshots = 3.

	Unique seeds			**Threat$_{IDs}$**		
	All	Elig.	(%)	All	Elig.	(%)
Benign	6,111k	509,700	(*8.3*)	-	-	
Malicious	1,783	236	(*13.2*)	151	60	(*39.7*)

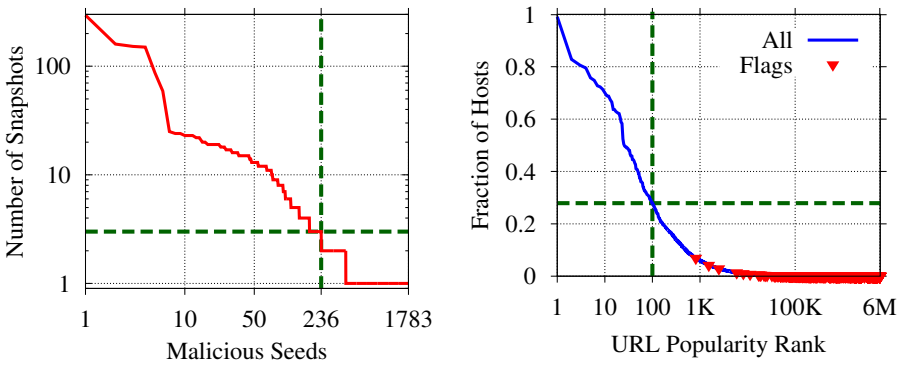
7.3.1 Events Eligible as Seeds

We here identify the amount of events eligible of becoming seeds. For more details about the data collection process and the available dataset, we remind to Sec. 6.1 and Sec. 6.2, respectively.

Recall that MAGMA's CG construction requires a recurrence of seed events over time and user population. The presence of recurrent events matches the basic properties of malicious activities, such as periodic reporting to the C&C center, or recurrent attempts to identify new victims. We also expect that malware creators try to disguise such repetitiveness as much as possible. In our data, indeed, we found 820 malicious hosts that had only one flagged event. If analyzed in isolation (on per host basis), these events would evade MAGMA's detection by not having any recurrence. Yet, by looking at correlation among different hosts, MAGMA is able to find commonalities between these events and tie them to a common malicious activity.

Fig. 7.5(a) reports the number of snapshots that can be associated to each malicious event. By considering 1,783 unique malicious events (see Tab. 6.1), we found that 236 unique seeds can be associated to at least three independent snapshots. Setting minSnapshots=3, these events become fully characterizable by MAGMA. Looking at absolute numbers, MAGMA can provide insights in 95% of malicious snapshots in our dataset, which is equivalent to 40k out of 42k flagged records (see Tab. 6.1). We also emphasize the diversity of MAGMA's characterization capabilities, noting that the events in scope correspond to 60 different types of threats.

In summary, Tab. 7.1 details the amount of events eligible to be seeds for both benign and malicious events when minSnapshots=3. We observe that only 509,700 (8.3%) benign events are repetitive enough to be considered by our system. This is largely expected: Benign traffic is mostly related to activities of human users, and would not access identical objects as recurrently as malware. To test our system, we next use all 236 malicious seeds and combine them with 664 randomly selected benign seeds.



(a) Snapshots per malicious events. (b) Popularity of HTTP objects.

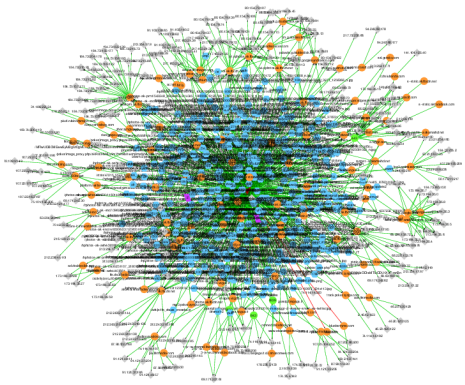
Figure 7.5: Dataset characterization. (a) Seeds generating at least 3 snapshots are processed by our system; (b) Top-100 HTTP objects are whitelisted.

7.3.2 Whitelisting

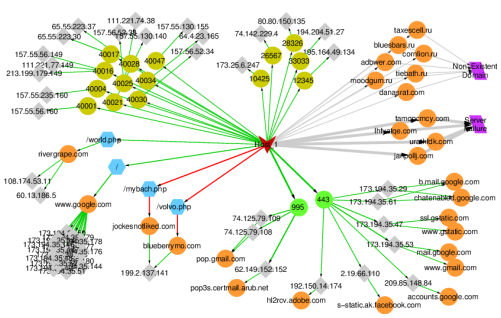
Fig. 7.5(b) shows the HTTP event popularity, i.e., the fraction of hosts that accessed a given URL (with stripped parameters). Note the log scale on x-axis. The figure shows the classic heavy-tailed popularity. Top URLs are clearly very common among most of the hosts. These include social network buttons (e.g., `www.facebook.com/plugins/like.php`), analytics services (e.g., `www.google-analytics.com/ga.js`), software update checks, etc. Red triangles highlight those events that are considered malicious by the IDS. The most diffused type of attack, i.e., a Drive-by Download threat, infects about 800 hosts (3.8% of hosts). The huge tail confirms the intuition that most of the URLs are accessed by few hosts only.

Leveraging the stealthy nature of malicious traffic, and thanks to the fact that few users are actually infected by a given malware, we adopt whitelisting as a common technique used to both reduce the amount of information to process and to discard data that would possibly pollute the analysis. We built a whitelist that targets very popular events, which adds little information or create noise. Instead of creating a manual list of popular events, we opt for a *dynamic* and *context-aware* approach. MAGMA builds a whitelist based on events popularity among clients and selects the top-k elements to be ignored during the processing. We whitelist single HTTP events and not the entire websites, as it is known that malware can be hosted and distributed also from benign services. We pick top most popular 100 HTTP events (highlighted in Fig. 7.5(b) by dashed green lines), i.e., we filter those events that are common to more than 23% of users. This is equivalent to assume that the most popular malware has infected less than 23% of the population.

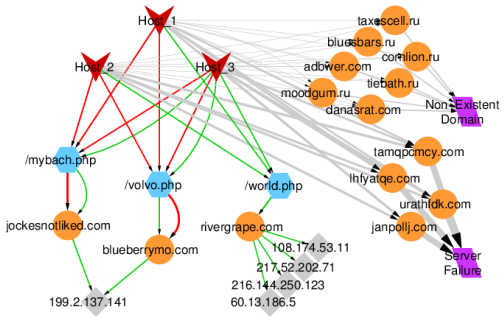
7.3.3 Connectivity Graph Construction Steps



(a) Initial graph from a single snapshot.



(b) Host-CG of a single client.



(c) Final Seed-CG.

Figure 7.6: Evolution of Network Connectivity Graphs at several steps of our methodology. The event under study `http://jockesnotliked.com/mybach.php` is reported as malicious by our oracle. Three clients are flagged for this event: Two generate two analysis snapshots each, while the third client generates eight snapshots.

In Fig. 7.6, we present an example of CG evolution according to the steps of the methodology described in Sec. 7.2. We consider the malicious seed `http://jockesnotliked.com/mybach.php`.

- Fig. 7.6(a) shows the network traffic produced by a host during a single 30-minute snapshot centered on the seed. Obviously, this graph is very difficult to interpret.
- Fig. 7.6(b) presents the Host-CG of one of the three clients (red marker in the center) involved in the malicious activity. Despite being already clearer and understandable with little effort, it still contains some nodes

related to ordinary user's behavior that are not part of the malicious activity. For instance, HTTPS and Post Office Protocol (POP3) Secure transactions on the TCP layer (light green circles in bottom-right part) are related to mail exchange and login to Google services, while flows over UDP (olive green circles in the upper part) are mostly directed to the Skype service.

- Fig. 7.6(c) corresponds to the final Seed-CG generated by our system when fusing the Host-CG of three different hosts. It is now much easier to identify the events involved in the suspicious activities. Events highlighted by red edges are those considered malicious by our oracle. The richness of the provided indications stems from the augmented context that we provide about these clients. First, the clients access three URLs (blue hexagons) hosted by three hostnames (orange circles) all of which now become an indication of a suspicious infrastructure. Next, two of the contacted hostnames (`jockesnotliked.com` and `blueberrymo.com`) use the same IP address (gray diamonds) suggesting for a potential obfuscation by hostname flipping and resources reusage, both common practices among malicious adversaries. The third hostname (`rivergrape.com`) is distributed over several mirrors whose IP addresses belong to very different subnets, a hint of cheap infrastructure or zombies that were previously infected by the malware. Finally, the right side of Fig. 7.6(c) shows another layer of information indicating multiple failures of DNS queries (purple boxes). This reaffirms our suspiciousness.

Apart from providing more context for the malicious activity, our system discovers new malicious objects and improves the flagging consistency of our oracle IDS. For example, the object `bluberrymo.com/volvo.php` is consistently included in malicious graphs, while the IDS occasionally misses it. Our system also discovered a new object `rivergrape.com/world.php`, and we confirmed its maliciousness across several other security tools such as VirusTotal [112].

7.3.4 Impact of Pattern Filtering

We study the volume of information that Seed-CG creation process extracts from single seeds. Tab. 7.2 shows the average number of nodes included in the final Seed-CG. Three sets of parameters are reported, from a very selective common pattern filtering such as c1 = {MinSupport=1, minPopularity=1}, which selects only the objects that appear in all snapshots and for all hosts,

Table 7.2: Average number of nodes with different MinSupport and MinPopularity thresholds. The snapshot size Δ is set equal to 30 min.

Type	**Malicious**			**Benign**		
	c1	c2	c3	c1	c2	c3
Object-path	6.6	14.9	2351.4	18.5	56.3	3320.1
Hostname	7.0	16.8	691.5	8.1	28.9	781.2
Server IP	19.9	95.9	3423.0	39.0	161.2	6260.2
Dst-port TCP	0.2	0.6	79.9	0.8	4.1	194.2
Dst-port UDP	2.0	27.8	1335.1	2.8	42.2	3129.5
DNS error	0.3	2.4	40.4	0.2	0.6	19.4
Total	36.0	158.4	7921.5	69.4	293.3	13704.6

c1 = {MinSupport=1, minPopularity=1}
c2 = {MinSupport=0.5, minPopularity=1}
c3 = {MinSupport=0, minPopularity=0}

to c3 = {MinSupport=0, minPopularity=0}, which instead merges and fuses all patterns independently of their support and popularity. c2 = {MinSupport=0.5, minPopularity=1} is the suggested default parameter setting.

Results clearly show that, starting from a single event, MAGMA builds graphs with hundreds of nodes. Note how the number of elements grows consistently when selecting less restrictive thresholds for MinSupport and minPopularity. The number of elements is indeed very large for c3, where no item is discarded and several thousands of nodes are retained and included in the CGs. This hurts the amount of information offered to the security analyst (see Fig. 7.6(a) for instance). c2 offers a good trade-off between descriptiveness and richness of the final CG. In addition, note that malicious Seed-CGs generally have fewer nodes, except for the nodes that represent DNS errors (last row of Tab. 7.2). For c2, the number of common object-paths (first row of Tab. 7.2) found in benign CGs is approximately four times larger than in malicious CGs, suggesting that benign Web pages are more complicated than malicious HTTP patterns.

Fig. 7.7 shows the average number of nodes in Seed-CGs according to the selected snapshot size, Δ, for the parameter set c2. As expected, CGs contain only the seed event and few other nodes when selecting small Δ, e.g., lower than 1 min. On the other hand, the number of nodes increases with the snapshot size, peaking at more than 200 nodes on average with a 60 min snapshot. Interestingly, the number of object-paths and hostnames

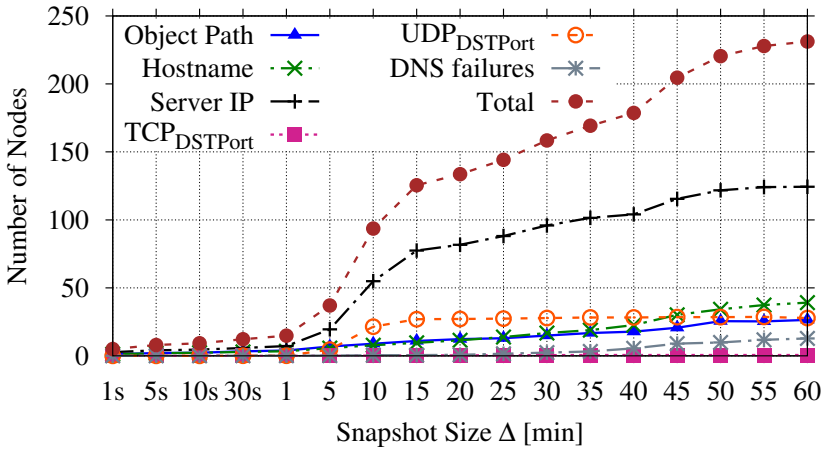


Figure 7.7: Average amount of nodes with different snapshot sizes. Parameters MinSupport and MinPopularity are set to c2 configuration.

does not increase dramatically with higher Δ, proving the effectiveness of the Common Patterns Mining technique. The only node type showing higher inflation is the Server IP address, owing to malicious activities poking benign infrastructures hosted on CDNs.

7.4 Examples of Connectivity Graphs

In the following, we provide some examples of CGs covering several typologies of malware found in our dataset, each presenting different behaviors and network patterns.

7.4.1 Cycbot Backdoor Activity

Cycbot is a backdoor trojan that allows cyber-criminals to access infected computers remotely. This causes victims' hosts to be exploited for large-scale attacks, and to potential leakages of personal sensitive information.

Fig. 7.8 shows the CG of the event `*/logo.png` for which our oracle raises an alarm. Interestingly, more than 80 hostnames seem to serve the malicious file `logo.png` (cloud of orange circles in Fig. 7.8). All these hostnames have a third-level domain name exhibiting random strings that are made of both characters and numbers, and are hard to code with regular expressions. This technique, known as *fast-flux*, allows attackers to hide malicious infrastructures by generating hostnames that are registered to the DNS and lately removed with a high frequency. This makes the detection harder, circum-

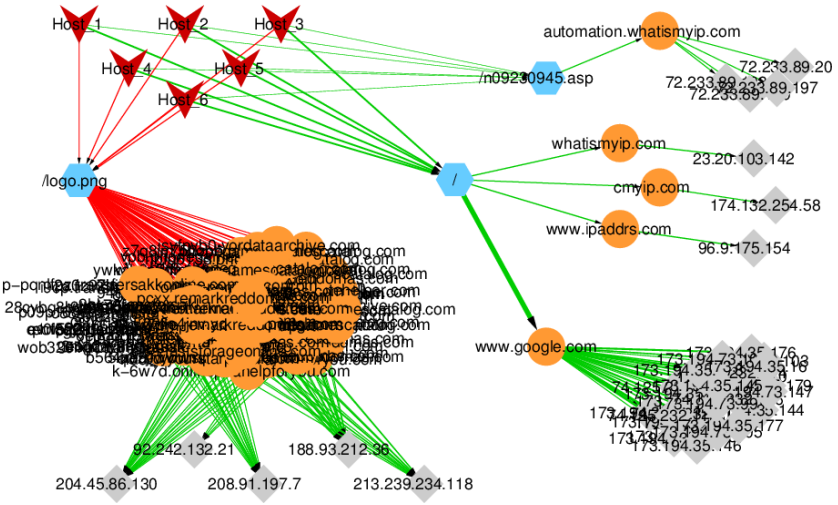


Figure 7.8: CG for Cycbot Backdoor Activity backdoor trojan. Notice the *fast-flux* domain name shuffling (orange circles in lower-left part), and the IP address detection Web pages (top-right) to assess victims' reachability.

vents blacklisting, and guarantees a longer reachability to the infrastructure. Considering only second-level domains, the number of hostnames drops to 10, each presenting an appealing name acting as a lure for potential victims, e.g., `faststorageonline.com`, `phonegamescatalog.com`, `wwwmp3archives.com`, etc. The entire set of domain names is hosted on few servers, heading to 5 IP addresses. These IPs are not organized in a structured CDN and do not belong to the same subnet, suggesting for the usage of infected machines scattered in the network.

The right part of Fig. 7.8 includes some benign objects. While these may seem false positives, this is not the case. Looking closer at the top of Fig. 7.8, it is easy to realize that contacted websites host services aimed at the discovery of the public IP address of the host. Such behavior is coherent considering the intent of the malware we are facing. Being a backdoor trojan, the infected client has to be reachable by the cyber-criminals, but connectivity issues, e.g., hosts behind NATs or firewalls, might preclude the reachability of the victim. Looking at the bottom-right part of the CG, we observe that the malware is checking Internet connectivity by visiting the `www.google.com` homepage, another test run by the malware to gather connectivity properties of the victim.

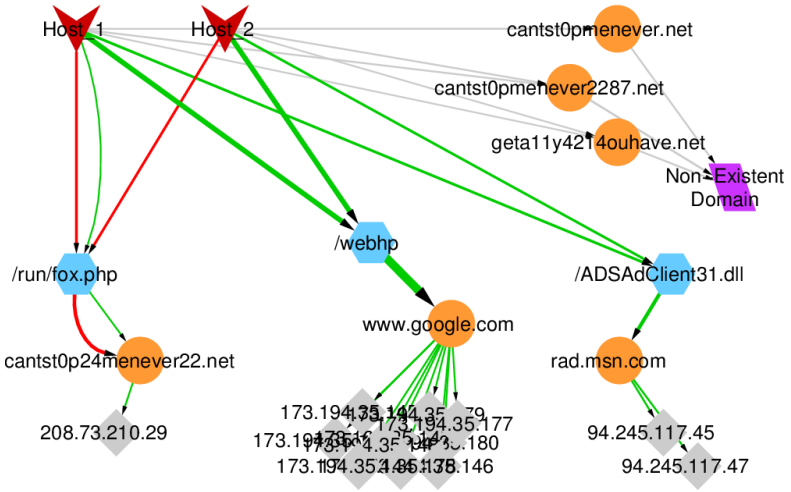


Figure 7.9: CG for Downloader.Dromedian Communication trojan. Notice the presence of legitimate nodes being contacted by malware.

7.4.2 Downloader.Dromedian Communication

Downloader.Dromedan is a trojan horse that runs silently on the victim's host, downloading and putting in execution additional threats. This malware connects to remote malicious infrastructures and C&C networks in order to fetch and execute additional malware and Potentially Unwanted Programs (PUPs). In most cases, it causes the redirection of ordinary Web surfing traffic to malicious websites through the installation of toolbars in Web browsers. Such toolbars force the user to visit certain content in order to generate profit for malicious attackers.

Fig. 7.9 depicts the CG derived from the analysis of the malicious event `cantst0p24menever22.net/run/fox.php`, related to Downloader.Dromedian. Two clients are infected and, in addition to the seed (red arrows), they both connect to `www.google.com/webhp` repeatedly. While the Google Web page is not malicious per-se, it is caused to frequently appear due to PUPs related to Conduit search hijacker. Several unwanted toolbars like Delta Search Toolbar, Social Search Toolbar, and Internet Helper Toolbar are related to Conduit search, and cause Google Webhp redirects by hijacking the browser settings. Looking at the top-right part of the CG, infected clients query three hostnames causing an error at the DNS resolver side, owing to the fact that the two exploited hosts try to connect to the remote malicious infrastructure. Our suspiciousness on such event is reinforced by the similarity of the queried hostnames with the successful one (i.e., `cantst0p24menever22.net`), and by

the presence of numbers interleaving characters with a random fashion, possibly trying to avoid blacklisting.

7.4.3 Mass Injection Website

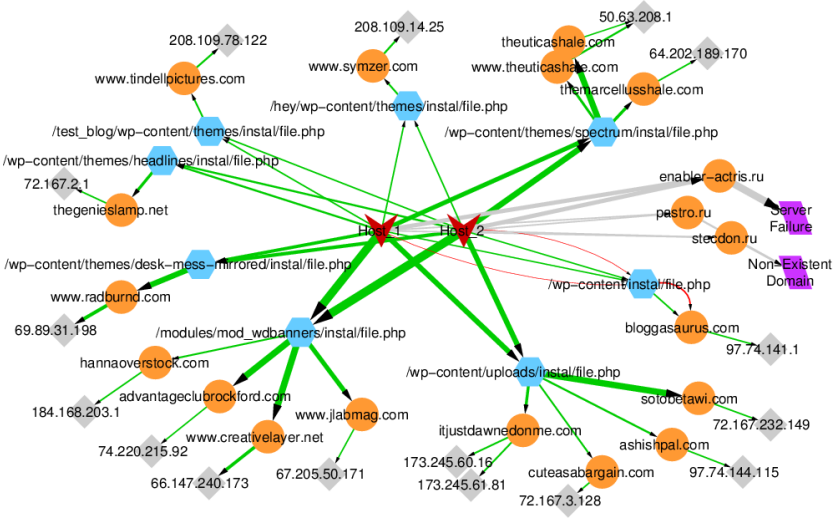


Figure 7.10: CG for Mass Injection Website attack. Notice all URLs containing "wp-content", hinting for an exploited WordPress vulnerability. Victims are forced in a redirection chain through websites hosting Exploit Kits.

The Mass Injection Website attack was detected for the first time in March 2011 and, differently from malware discussed above, does not target the host of the victim directly, but leverages vulnerabilities of legitimate websites to inject malicious scripts and hidden `iframes`. In turn, users visiting compromised websites are victims of a redirection chain that forces them to visit third-party websites hosting EKs. Such EKs target potential vulnerabilities at the client side so that once the victim lands on the effective EK, her machine gets infected as well.

Fig. 7.10 shows the network behavior of two hosts being victims of the Mass Injection exploitation. Our oracle considers malicious only the seed `bloggasaurus.com/wp-content/instal/file.php` but our CG is populated with other URLs all terminating with `/instal/file.php`. Interestingly, many of these URLs also include the substring *wp-content*, suggesting for an exploited vulnerability in the WordPress blogging tool. Our suspiciousness is confirmed by the fact that both victim hosts exhibit the same identical behavior towards all the URLs depicted in the CG. The volume of issued

requests is identical, and, considering the contacted URLs in a temporal sequence, it is possible to clearly spot repeated patterns over time, i.e., both hosts visit other URLs in a deterministic way. Moreover, such Web pages host different kinds of content, ranging from newscast to music festivals and healthcare. Thus, it is almost impossible that two users visit the same pages, the same number of times, in the same order. This confirms the ongoing automated Web scan the user is not aware of.

7.4.4 Overall Analysis of CGs

We now offer a thorough analysis of URLs MAGMA identified as belonging to CGs. In particular, for each of the 236 malicious seeds, we extract the corresponding CG. Overall, 5213 unique items appear, out of which 2393 are HTTP requests that the IDS oracle does not flag. For each of these requests, we investigate if they are malicious or benign. For this purpose, we use again the online service VirusTotal [112]. In addition, we also double check each of them using Snort [36]. Results show that 1580 (66%) of the discovered items are labeled as malicious by either VirusTotal or Snort, thus confirming the items in the CGs generated by MAGMA form a better and more complete picture than the one originally offered by the oracle.

CGs also include 1114 benign objects that no IDS or antivirus flags. While these may seem false positives, it is often not the case. For instance, we have seen perfectly legitimate URLs being included in a malicious graph. We have already verified that benign URLs are indeed part of malicious activities that run checks (e.g., the `www.google.com` or `whatismyip.org` for the Cycbot case, Fig. 7.8). Furthermore, infected hosts often contact legitimate servers to run DDoS attacks, or to generate fake clicks on advertisements, or in general to spread the malware and run attacks. IDSes cannot flag these events as malicious, since the corresponding services are not malicious. In contrast, MAGMA is capable of discovering such interactions between malicious and legitimate infrastructures, and to expose them to the security analyst.

7.4.5 Contrasting against Benign CGs

As a point of comparison, we show the CG produced at the end of the process triggered by a benign seed, `img.alice.org/tracks/bi/images/bi_clk.gif` in Fig. 7.11. If paired with the malicious CGs shown above, it looks very different. In the bottom part, there is a legitimate CDN (i.e., the Facebook CDN) serving content from 18 IP addresses that are organized in three /24 groups of six IPs each. In addition, multiple HTML files are retrieved from

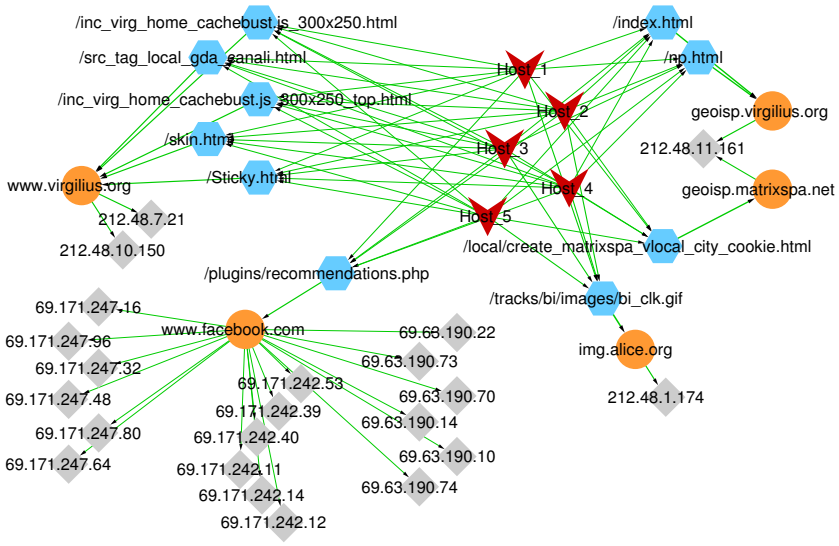


Figure 7.11: CG for a random benign seed. Notice the presence of structured CDNs and the download of multiple objects from the same domains, suggesting for a legitimate Web page being fetched.

the same domain name (top-left part), a common practice to fetch all the content needed to render a legitimate Web page.

Such observations hold for other benign CGs as well. They typically show the absence of failed DNS requests, the availability of multiple files and file types on the same domain name, the presence of structured IP address sets or CDNs hosting content, the absence of failed HTTP requests, etc.

7.5 Conclusions

In this chapter, we presented a system able to identify network events correlated with malicious traffic. Starting from a malicious seed, i.e., a network activity for which a reference IDS triggers an alarm, our system leverages both spatial and temporal recurrence of events and frames them in a Connectivity Graph (CG), that is a focused representation of the malicious activities over multiple network layers. Our system delivers an enriched set of network events related to the malicious seed and is capable of increasing the knowledge of the incident.

We proved our approach is effective against different classes of malware, each showing peculiar behaviors and network patterns. In all cases, our methodology provided a rich and interpretable characterization of the malicious activity, facilitating the understanding of malicious attacks and sup-

porting the forensic investigations of the security analyst. In addition to this, the information contained in the produced output can be used to train classifiers able to distinguish between malicious and benign CGs. Through the extraction of signatures, it would also be possible to spot new malicious activities not known yet, i.e., zero-day malware, improving the detection rate of current security solutions, and warning the security analysis on targeted suspicious activities.

Chapter 8

MAGMA

Supervised Classifier

In this chapter, we present the design of a classifier that aims at detecting any generic malicious pattern. We build on top of the focused and rich description of network activities provided by the CGs previously introduced to extract representative features and train supervised classifiers consequently. The goal is to design a classifier that is able to answer the question "Is the Connectivity Graph being analyzed representing malicious or benign activities?".

Following the best practices from the machine learning community, we consider multiple classifiers and accuracy assessment techniques to run a thorough evaluation of the classification results. We also consider various typologies of features and run feature selection algorithms to identify the most relevant ones. Despite the heterogeneity of both malicious and benign patterns, the MAGMA classifier is able to achieve accuracy figures as high as 95%. Moreover, it shows to be very robust with respect to parameters settings in the CG domain.

8.1 Overview

Fig. 8.1 shows the processes needed to train a classifier able to distinguish between malicious and benign Seed-CG. As first step, we use the alarms raised by the commercial IDS to build two distinct sets of Seed-CGs, one containing CGs generated by malicious seeds, the other containing CGs of benign seeds. We then leverage the descriptiveness of Seed-CGs to define a set of features with which the classifier can be trained and tested. These include: (i) Graph topology properties (e.g., number of nodes, node degrees); (ii) HTTP header parameters (e.g., response status, distinct user-agents,

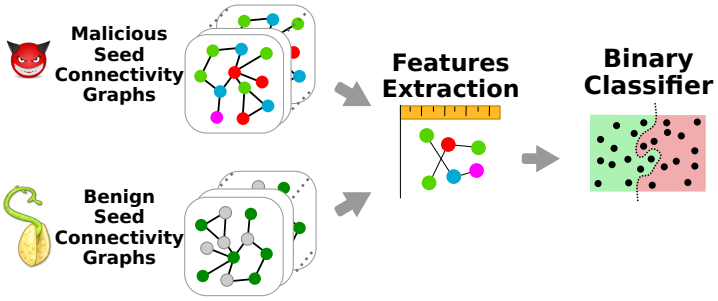


Figure 8.1: MAGMA classification process overview.

content-type); (iii) syntax properties extracted from the names (e.g., string length, number of subdomains, digit–characters ratio); and (iv) occurrence properties (e.g., minimum, maximum, average recurrence of specific events). Some of these features are driven by the domain knowledge, while others are generic and considered to avoid biasing towards a specific threat. The complete set of features is described in App. A.

As last step, we run an exhaustive set of experiments to assess the accuracy of the classifier under a variety of conditions and its sensitivity to parameter setting. The results is MAGMA, a behavioral classifier able to label Seed-CGs as malicious or benign with high accuracy.

8.2 Classifier Choice

We now design a supervised classifier and train it using the labeled dataset of graphs obtained considering malicious and benign seeds. We consider different decision tree classifiers: (i) The original J48 (an open source implementation of the C4.5 decision tree); (ii) bagging coupled with J48; and (iii) Random Forest (RF). Decision trees are known for being scalable and offering interpretative classification models [109].

In a decision tree, internal nodes represent tests on individual features, each branch is an outcome of the tests, and each leaf node represents a decision, i.e., a class label. The paths from the root to a leaf represent classification rules. Bagging is a process that improves stability and accuracy by training m decision trees on m independent samples of the training set. The m models are combined by voting at the end. Random Forest is an extension of the bagging process such that, at each candidate branch in the learning process, a random subset of the features is selected to avoid strong features from biasing the construction of trees.

Since MAGMA aims at the classification of malicious patterns in general and does not target a specific class of malicious behaviors, we define an extensive set of features and extract them from Seed-CGs. Four different domains are covered:

- **Graph topology**, e.g., the number of distinct nodes for each type, min/max/avg/std of in-degree and out-degree for each node type, graph giant connection ratio, etc.;
- **HTTP**, e.g., the number of GET/POST events, min/max/avg/std of the length of user-agent string, etc.;
- **URL syntax**, e.g., the number of hostname accessed directly using the IP address, or starting with `www`, etc.;
- **Occurences**, i.e., min/max/avg/std of the number of events for each node type.

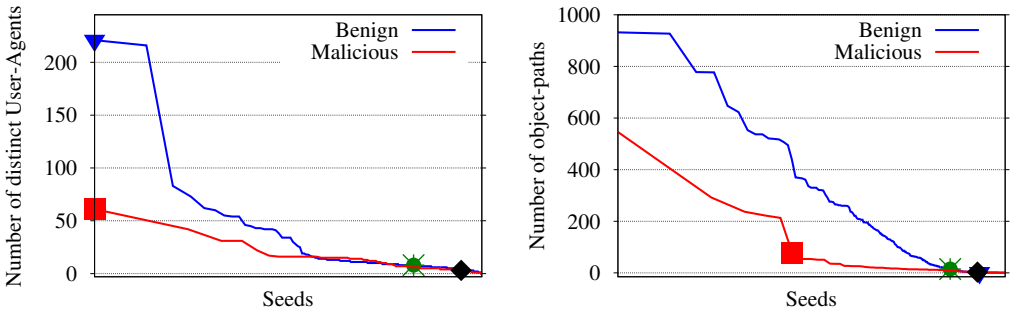
The choice of features is partly driven by domain knowledge or has been previously used in the literature. Some features are instead generic but could be useful in making the distinction. In total, 111 features are extracted from each CG, as detailed in App. A.

In the following, we consider the classifiers trained using (i) all; (ii) only HTTP; (iii) only Topology; and (iv) only Syntax features. We do this in order to compare against the previous works in which only HTTP or syntax has been used to target a specific malware. For comparison, we consider the subset of features suggested by the mRMR feature selection algorithm [48] (the selected features are reported in bold characters in Tab. A.1 in App. A). Note that mRMR selects features from all four of our domains.

8.3 Feature Characterization

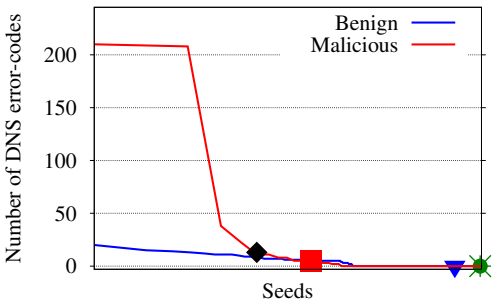
Tab. 7.2 hints that CGs obtained from malicious and benign seeds contain a different amount of nodes and edges. Here, we briefly illustrate the extent of feature differences extracted from CGs. Fig. 8.2 compares the number of occurrences of three features in individual CGs: (i) Number of distinct User-Agents; (ii) number of object-paths; and (iii) number of DNS failures. Two benign and two malicious seeds are highlighted for comparison.

Consider first the number of distinct User-Agents (Fig. 8.2(a)). The intuition is that malware could abuse the semantic associated to the User-Agent information and generate a large number of semi-random strings. This is what happens with `http://badi4net.no-ip.org/realtime.xmltmp` (red



(a) Number of distinct User Agents.

(b) Number of object-paths.



(c) Number of DNS error-codes.

Figure 8.2: Feature distributions for malicious and benign CGs. Due to the variety of patterns, the usage of simple heuristics is not suitable to separate benign and malicious CGs.

square), which is a malware that generates click-fraud and "impersonates" different browsers. Surprisingly, benign applications also abuse the User-Agent field, e.g., `liveupdate.symanteclive update.com` (blue triangle) encodes the update versions in user agents, generating more than 200 agents.

Looking at the number of HTTP objects, Fig. 8.2(b) confirms the intuition that benign patterns include more objects. Yet, there are some malicious patterns that have a large number of objects and a lot of benign CG have few HTTP objects. This is the case of the previously investigated seed in Fig. 8.2(a) (black diamond and green star). Finally, look at Fig. 8.2(c), which represents the number of DNS failures. Also in this case we expect a high number of failing DNS requests to be characteristic of malicious activity, but there are many benign CGs exhibiting a similar number of failures.

Two conclusions can be drawn from these examples. First, the 60 threats in the dataset that we use exhibit a wide range of patterns. Second, there are no easy means to separate malicious and benign CGs using some simple

heuristic. A state-of-the-art classifier is needed to combine different planes of information and make the distinction.

8.4 Classification Results

To assess the performance of MAGMA, we follow best practices suggested by the machine learning community. We consider a labeled dataset, where ground truth labels are provided by the IDS and then we train and test performance using this labeled dataset.

8.4.1 Cross-validation and Performance Metrics

We split the labeled dataset of N eligible seeds in two parts, one for training the classifier, and the other for testing its performance.

We employ several validation methodologies: (i) 66% split; (ii) k-fold cross-validation; and (iii) leave-one-out cross-validation methodologies. In the first methodology, we run a single experiment using 66% of our dataset for training and the remaining 33% for testing. k-fold cross validation generalizes this such that k equal size subsets are randomly generated. Then, k experiments are run, where $k-1$ subsets are used for training, and the remaining 1 is used for testing. The results are computed over all k runs. Finally, "leave-one-out" is an exhaustive cross-validation methodology in which, for each of the N elements in the labeled dataset, $N-1$ are used for training, and 1 is used for testing. The results are then computed over N independent experiments. Exhaustive cross-validation methods are preferred since they learn and test all possible ways to divide the original sample into a training and a validation set. They are considered to be the most accurate means of testing a classifier, but they require a very large number of tests. For $N \approx 900$, we can afford the complexity of the leave-one-out in the following.

We measure the performance in terms of *accuracy*, i.e., the fraction of valid results over the number of tests. We also report the confusion matrix which details the number of true positives and false negatives for each class, i.e., for malicious (benign) seeds, it shows the number of events that were correctly classified as malicious (benign) and the number of events that were erroneously classified as benign (malicious).

8.4.2 Classifier and Feature Impact

We start by evaluating MAGMA with the default parameter setting, i.e., $\Delta = 30\,\text{min}$, MinSup=0.5, and MinPopularity=1. Fig. 8.3 reports the classification

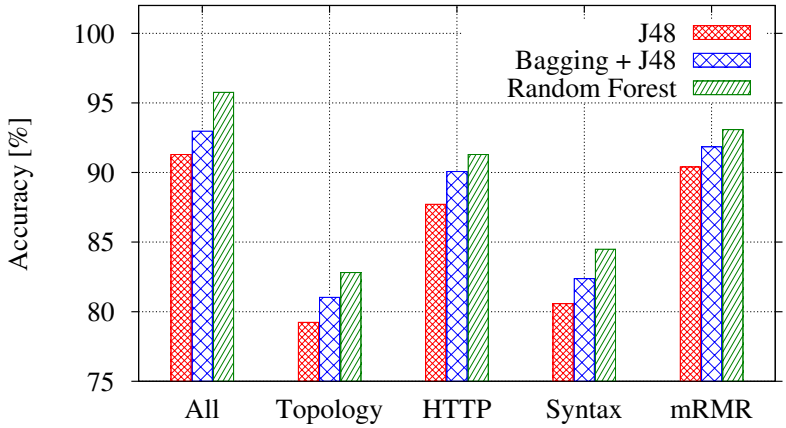


Figure 8.3: Comparison of MAGMA's classification accuracy for different classifiers and set of features.

Table 8.1: Confusion matrix for Random Forest classifier and all features.

	Predicted Class	
	Malicious	Benign
Malicious	218	18
Benign	21	643

accuracy for the three classifiers, considering different sets of features (see Tab. A.1 in App. A). Observe how Random Forest consistently provides the best results, with J48 providing the worst. Accuracy is higher when all features are used, with all subsets contributing to improving performance. The two observations confirm the richness of information offered by Seed-CGs, as well as the need to consider a wide range of generic features that do not focus on particular types of malware. We also confirm this reasoning via mRMR checks, where only few features are selected, but all of them are always from different domains.

When all features are offered to the Random Forest classifier, accuracy exceeds 95%. This is an excellent result in general, which is confirmed by the confusion matrix reported in Tab. 8.1. The rows of the matrix specify the ground truth class, while the columns indicate the predicted class. Cells on the diagonal represent the number of true positives, while cells outside represent the false positives (or false negatives). The results confirm that the recall and precision in pattern classification are very high for both classes.

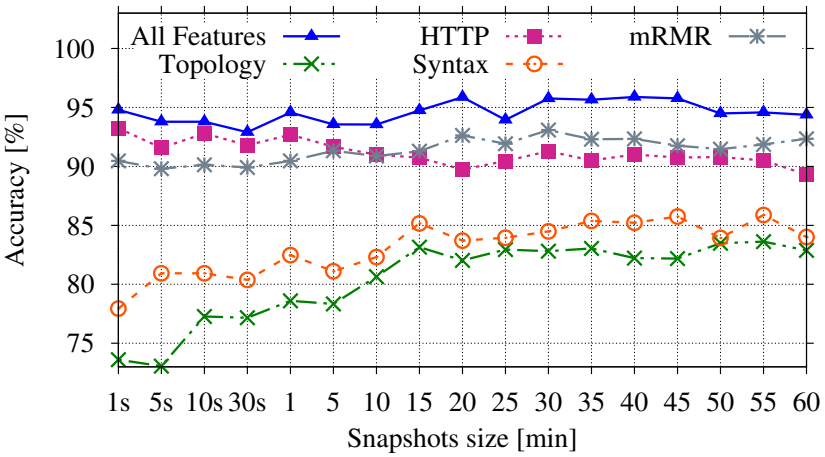


Figure 8.4: Classifier accuracy depending on the observation snapshot duration Δ with Random Forest classifier and different feature combinations.

8.4.3 Parameter Sensitivity

We now focus on the impact of parameter setting. Fig. 8.4 shows the impact of the choice of Δ, which we vary from 1 s to 60 min. It considers only the Random Forest, with MinSup=0.5, and MinPopularity=1. The experiments are repeated for different feature combinations. Results show that Δ has a limited impact. The intuition is that typical activity related to an event lasts few seconds during which the application running at the host generates a burst of events. Only for very small values of Δ indeed it is possible to appreciate a generic decrease of accuracy due to a limited number of events that fall within the snapshots. Interestingly, we observe that HTTP features tend to be more significant for small values of Δ, while graph topology features gains of importance for larger values of Δ. The drop of accuracy for a HTTP only based classifier is due to noise infiltrating into the benign graphs when large snapshots are considered. At the same time, the rich graphs are better characterized by topology features. Notice how the classifier trained considering all features is able to trade the drop of HTTP feature information with the increase of information offered by other features.

With $\Delta = 30$ min, we change the MinSup, and MinPopularity parameters. In Sec. 7.3.4 we already detailed how a different choice of parameters induces on the filtering and enrichment process (see Tab. 7.2). We now compare the impact on classification accuracy. Recall, that the number of snapshots and of hosts presenting a seed is rather limited. As such, we can only coarsely choose the parameters. Fig. 8.5 reports results. We observe that, also in this case, the impact of parameter setting is not crucial. However, by applying a

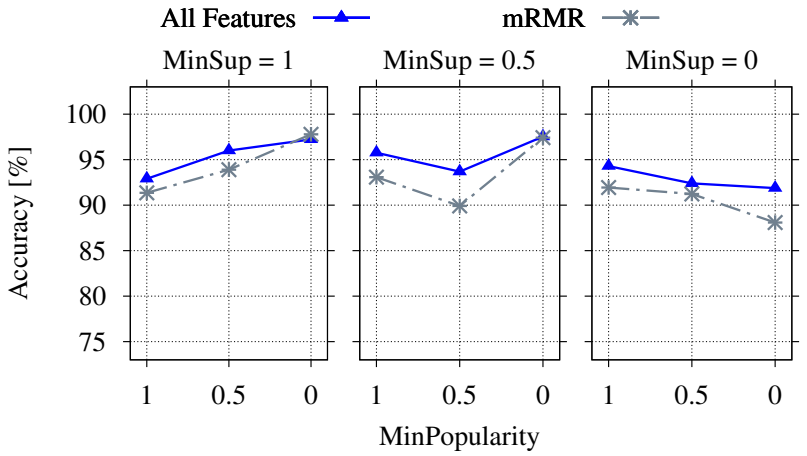


Figure 8.5: Sensitivity analysis on MinSup and MinPopularity with Random Forest classifier, all and mRMR features, $\Delta = 30$ min.

too selective choice, e.g., MinSup=1, and MinPopularity=1, or a too permissive filter, e.g., MinSup=0, and MinPopularity=0, the accuracy tends to decrease. In the first case, too few events are left in the common pattern extraction. In the second case, too many events are instead accepted so that CGs appear to be noisy. Trading between minimum frequency in snapshot and minimum frequency among hosts provides a good trade-off.

Notice that the choice of MinSup, MinPopularity impacts also the number of events that appear in Seed-CGs, which is then offered to the security analyst in case he/she likes to investigate a decision returned by MAGMA. The more restrictive they are, the smaller the number of events. The choice of MinSup=0.5 and MinPopularity=1 results in a good balance between accuracy and richness of the graph, as depicted in Fig. 7.6.

8.4.4 Additional Experiment

We now aim at assessing the usage of MAGMA in the wild. To this extent, we run an experiment on a separate trace which includes only traffic from 15 hosts monitored for one day. The IDS does not flag any of the events appearing in the traffic, therefore these hosts should be considered as not infected by any malware. Overall, 43,000 distinct HTTP events are collected, of which 1868 are seen at least 3 times, and thus are valid seeds.

For each seed, we extract the CG and run it through the MAGMA classifier that we previously trained. We let then MAGMA classify each of the CGs. It returns 1852 benign tests, and only 16 malicious tests. By consid-

ering the oracle labels, the former have to be considered as true negatives (i.e., benign events classified as not malicious). The latter instead have to be considered as false positives (i.e., benign events misclassified as malicious). This corresponds to 99.14% of accuracy, with a mere 0.86% of false positive.

We further investigate the 16 misclassified cases using again Snort and VirusTotal. Snort detects 10 malicious events out of the 16 misclassified cases, while VirusTotal raises 4 warnings for events that were already flagged by Snort. Cross-checking VirusTotal threat descriptions and Snort detection-rules documentation, it appears that the detected events are related to *Simbar Spyware*, a redirecting toolbar affecting Internet Explorer, *Sgrunt Dialer*, a Trojan virus that limits the access to files and programs, and *AskSearch Toolbar* that is responsible for inflating clicks on advertisement to monetize traffic. In a nutshell, MAGMA identified 10 threats that the IDS ignored (but other tools have signatures for). This further reduces the false positive to only 6 cases over 1868 tests (0.3%).

8.5 Conclusions

In this chapter we leveraged the representativeness of Seed-CGs to train a classifier and distinguish between CGs belonging to the benign or to the malicious class. We do so by extracting features covering multiple aspects of CGs, including graph topology, HTTP header fields, URL syntax and occurrences properties. We use such features to train a supervised classifier following the recommendations of the machine learning community and including the usage of feature selection algorithms.

We demonstrated the effectiveness of the classification process considering multiple classifiers, cross-validation techniques, and parameter setting applied to the source CGs. The achieved results are more than satisfactory, with a classification accuracy exceeding 95% in the best scenario and never falling below 90% for the Random Forest classifier.

We also proved the efficacy of the extracted model on a test set made of 1868 benign CGs that we here classified. Results showed that very few of them were misclassified as false positives and, in some relevant cases, such false positives were actually representative of malicious activities according to other security tools, i.e., VirusTotal and Snort, being thus true positives.

Summary

In this part of the thesis, we tackled the problem of malware: Unwanted traffic that arms users' privacy and data confidentiality. To fight against this increasing threat, we collected real traffic from a large operational network and characterized the behavior of malicious activities. We then presented MAGMA, an automatic system for the identification of malicious activities.

MAGMA starts its process from simple events (e.g., HTTP requests, DNS queries, etc.) and leverages both their spatial (over multiple hosts) and temporal (over time) recurrence to depict an accurate picture of related activities. The outcome is an enriched and concise representation of the event that triggered the process (i.e., the seed) framed with highly correlated events that are part of the same activity.

In a first stage, MAGMA presents the output of the process by means of a Network Connectivity Graph, a multi-layered representation of the identified events where each layer is representative for a specific network protocol, e.g., DNS, TCP, UDP, etc. At a later stage, a decision tree classifier is trained on a set of graphs produced by benign and malicious seeds. Seeds are known to be either benign or malicious thanks to the labels provided by a reference IDS. The set is made of tens of different threat types, enabling the training phase to be as broad as possible given the heterogeneity of considered activities.

MAGMA, thus, results in a general purpose malware classifier, able to leverage common features in the network traffic that characterize different families of malware. It proved to have a classification accuracy as high as 95%, with its performance being consistently satisfactory and resilient to parameter settings. We acknowledge that MAGMA applicability is limited only to threat families exhibiting recurrent patterns over time and multiple hosts. However, we proved that MAGMA is able to effectively frame malicious activities in a detailed yet compact picture of undoubted value for the security analyst, as well as to distinguish with very high accuracy between benign and malicious activities. MAGMA is intended to facilitate the identification of previously unknown malware and to support forensic investigations by providing a rich and interpretable characterization of the malicious activity.

Part III

User Quality of Experience

Chapter 9

Web Quality of Experience

9.1 Introduction

In today's Internet, Web is still playing an important role among the plethora of services that can be accessed through the network. The browser has become the means to access a humongous amount of information, from Web searches to business applications, from personal communications to social networks and entertainment portals. Most of the communication at the application layer is still carried over HTTP/1.* (H1), a protocol that was defined at the end of last century and left almost unchanged since then. H1 is now called to accomplish tasks for which it was not designed for and faces complex network infrastructures including middle boxes (e.g., firewalls, Intrusion Detection System, and Carrier Grade NAT devices) that were not deployed at the time of its specification. As a result, H1 is becoming the de-facto "thin waist" of the Internet [19, 113], with security features offered by HTTPS that are also gaining momentum [10, 114].

Only recently a number of new protocols, namely HTTP/2 (H2) [115], SPDY - An experimental protocol for a faster web (SPDY) [20] and Quick UDP Internet Connections (QUIC) [22], have been proposed and are likely going to change the Web status quo. We are therefore facing important evolutions and having reliable ways to compare protocol performance becomes crucial before massive deployment of new protocols can take place [116].

However, measuring users' WebQoE is a challenging problem. Page complexity has grown to include hundreds of objects hosted on different servers, with browsers opening tens of connections to fetch them, and executing JavaScript code as part of the page construction process. A number of studies pointed out the importance of delay and of its direct relationship to the value of business. For instance, Amazon and Google report losses in the

0.6-1.2% range for delay increasing by 0.4-1 sec [117], while Shopzilla [118] reported +12% revenue for a 5 sec reduction in the time needed to load the page, achieved through a major redesign of the website.

The hidden correlation between these factors is the impact that delay has on the WebQoE: The higher the delay, the lower the WebQoE, the worse the experience, the higher the likelihood of user disengagement, the larger the economic losses. While the existence of a relationship between delay and WebQoE is beyond any doubt, it is more difficult to precisely pinpoint the delay of "which" event is the most important during the lifetime of a Web page, and, in turn, to understand how it impacts WebQoE.

Besides considering delay at application layer, i.e., relying on metrics belonging to the browser scope, it is worth focusing on external factors that might impair WebQoE. These can reside in the transport network and, more specifically, can be caused by middle boxes being deployed by ISPs. In recent years, indeed, ISPs started to use Carrier Grade NAT (CG-NAT) technologies to disguise their networks in large private sub-networks by assigning customers' routers private IP addresses, significantly reducing the amount of required public IP addresses and, in turn, the costs of maintaining them.

In a nutshell, a router implementing NAT functionalities remaps the IP address space of a private network into one (or more) public IP address(es). CG-NAT technologies extend this concept by masking a whole ISP network using NAT [119]. When communicating with hosts in the public Internet, the CG-NAT router temporary maps the private, edge-facing IP address of the customer to one available public, Internet-facing IP address.

However, NAT and CG-NAT middle boxes break the end-to-end paradigm of the Internet communication model. On the one hand, NAT-ed hosts cannot be directly addressed from the Internet. On the other hand, the NAT mapping operations may add delay to packets or cause losses. CG-NAT technologies can then have an impact on the time needed to fetch Web content, finally impacting users' experience.

9.2 Limitations of Current Practices

The state of the art in WebQoE evaluation has its roots in several metrics used for performance assessment, which can be classified in two main groups:

1. **Objective measurements**, e.g., the time at which the first Byte is received (TTFB), the time at which the first object is printed on screen (TTFP), the time needed to parse the Document Object Model (DOM), etc. Above all, the Page Load Time (PLT) [120], also known as onLoad, is the de-facto benchmark metric used for comparison, both

in research [121, 122, 123, 124] and in industry [125, 126, 127]. For instance, Alexa [125] reports the PLT exposing quantiles of the delay, while Google uses PLT to rank search results [126], although with a small weight [128].

2. **Subjective measurements**, i.e., the Mean Opinion Score (MOS), allow one to completely measure the WebQoE. However, it is extremely complex and expensive to run large MOS measurement campaigns. As such, automatic approaches to better capture the actual WebQoE have been proposed. These include the Above-the-fold [129] and the SpeedIndex [130], both introduced by Google in 2011 and 2012, respectively. Unfortunately, the relationship between such automatic measurements and the actual user experience has yet to be proven, and their computational complexity makes them difficult to use in practice.

All objective metrics elect a single marker as unique indicator of WebQoE. From the introductory discussion, however, it appears obvious that no single event can express all sort of intricate dependencies [123] between the download of required resources, the rendering process, and the user experience. As such, objective metrics can not fully reflect users' QoE in the complex "waterfall" [131] of network and browser events taking place during the page construction processes.

Subjective metrics are closer to the user perceived WebQoE and, in the case of MOS, they are fully representative of it. However, they are of little practical usage for large-scale measurements or automated tests as either they have a high computational complexity, which relegates them to laboratory experimentations, or they directly require users' intervention, which mandates the availability of volunteers and the execution of experiments in a controlled and repeatable scenario.

For what concerns network related factors, i.e., the deployment of CG-NAT devices, a significant effort has been put in standardizing CG-NAT technologies [132, 133] and traversal techniques [134, 135]. However, little is known on how and to what extent the traffic exchanged with the Internet is affected by such middle boxes and, consequently, if users' experience is negatively impacted.

9.3 Contributions and Work Organization

Recognizing intrinsic limits of metrics currently used for WebQoE estimation:

- We provide a complete taxonomy of the existing WebQoE metrics and tools, and introduce a generalization of Google's SpeedIndex. We par-

ticularize two new metrics providing a significant semantic relevance at a much lower computational cost;

- we present a comprehensive illustration of all WebQoE metrics on the top-100 Alexa Web pages, further elucidating relationships among metrics. Notably, we show that (i) an indetermination principle emerges when evaluating the SpeedIndex as its computation alters the very same experiment and that (ii) our proposed metrics remain highly correlated to the SpeedIndex despite their simplicity.

To assess how WebQoE is actually perceived by end users:

- We engineer a methodology comprising the design of a testbed to reproduce real Web pages in a controlled fashion. We leverage such testbed to collect volunteers' feedback in a controlled environment, where users are asked to access actual pages while we control network (i.e., latency and loss conditions), protocol (i.e., H1 vs H2) and application setup (i.e., domain sharding);

- we collect a dataset comprising over 4,000 samples of subjective feedback (i.e., MOS points), augmented with all objective measurements extracted automatically from the testbed;

- we dig into the data to shed light on actual user experience improvement when using H2 vs H1. Advantages appear to be mitigated with respect to those shown by objective metrics: Users report no differences in over half of the cases, and H2 is reported to improve WebQoE in 22% of cases only;

To fill the lack of discussion about the impact of CG-NAT on user experience:

- We define analytics to assess statistical differences between populations of users with either private or public IP addresses. For the purpose, we consider Key Performance Indicators (KPIs) that are relevant for user QoE, such as connection setup time, time needed to receive the first byte of useful payload for Web traffic, etc.;

- we apply these analytics to assess the impact of CG-NAT from an actual ISP deployment, processing the data collected by monitoring about 20,000 residential customers for one month. Results show that no statistically significant difference can be observed between the two populations for the considered performance indicators.

This part of the thesis is organized as follows:

Related work for both WebQoE metrics, MOS collection and analysis, and the impact of CG-NAT on Web traffic is presented in Sec. 9.4).

Chap. 10 starts tackling the problem of WebQoE estimation: (i) Sec. 10.1 reports a complete overview of metrics currently used for WebQoE assessment, generalizing Google's SpeedIndex and introducing two new metrics, which keep a high relevance at a much lower computational cost; (ii) Sec. 10.2 characterizes the considered metrics performing experiments on the top-100 Alexa websites, presents a correlation analysis among metrics, and highlights how the evaluation of computationally demanding metrics (e.g., the SpeedIndex) can affect the very same experiment.

It then continues moving the focus on the collection and the analysis of users' feedback, i.e., MOS grades. (i) Sec. 10.3 presents the testbed designed to execute the experiments detailing the Web pages considered, the engineering on the testbed on both the server and the client side, and the way MOS evaluations are collected; (ii) Sec. 10.4 presents the results obtained on a toy page where H2 is expected to outperform H1; (iii) Sec. 10.5 presents results with real Web pages, considering both objective metrics and the MOS for contrasting H1 vs H2, and shows the impact of deployment parameters such as latency diversity and domain sharding.

Chap. 11 describes CG-NAT devices deployed in ISP networks and their impact on Web traffic. (i) Sec. 11.1 presents the methodology used to extract meaningful analytics from raw traffic and process them for the assessment of CG-NAT impact; (ii) Sec. 11.2 introduces the statistical techniques used to quantify the distance between empirical probability distributions belonging to two populations of users, namely the ones provided with a public/private IP address; (iii) Sec. 11.3 identifies the performance indicators relevant for the purpose and defines metrics with which the two populations of users are contrasted; (iv) Sec. 11.4 presents the results for Web traffic.

Finally, App. B.1 better details the characteristics of Web pages used as catalog for the collection of MOS grades, while App. C.1 reports additional results related to CG-NAT technologies, e.g., their impact on P2P traffic, implicit filtering of unsolicited network activities, and potential resource saving for different NAT policies.

9.4 Related Work

9.4.1 WebQoE Experimentations

Since the original SPDY proposal [20], ended with the standardization in H2 [115], and the appearance of QUIC [22], researchers have been devoting increasing attention to the benchmarking and optimization of these protocols [116, 121, 123, 124, 136, 137, 138, 139, 140]. Tab. 9.1 (presented at the end of this section) surveys related work by describing the experiment scale, the testbed setup (i.e., servers, clients and network configuration), the set of pages considered (i.e., the page catalog), and the collected metrics. The picture is rather scattered and reflects the large space of options faced.

Experiment scale.
In terms of experiment scale, works collecting objective metrics span from several thousands (active testbeds [121, 123, 124, 136, 137, 138]) to several millions points (crawling [116] and server logs [139]). Conversely, studies employing actual user feedback (to the best of our knowledge, only [140] besides this work) are inherently of smaller scale.

Testbeds.
Testbed setups are either based on proxies [121, 136] or, as in this work, on locally controlled servers and networks [123, 124, 137, 138]. Few works leverage actual H2 servers in the Internet [116] or large corporate server logs [139]. Google Chrome is the most popular Web browser, although its version, the OS/device on which it runs (e.g., android mobile [136]), or the configurations (e.g., TLS disabled [138]) are not consistent across setups. It is also possible to find custom client implementations [123], or mixture of clients [139]. As for network setup, both controlled [123, 124, 137, 138] and uncontrolled [116,121,136] accesses can be found, including 3G/4G networks.

Page catalog.
In terms of pages used for testing, we observe a mixture of approaches. Alexa ranking is a popular source for the selection of websites. The number of sites ranges from 20 to 500, and page selection criterion (e.g., landing [136] vs non-landing [121]) differs. We also use Alexa to bias our choice towards popular websites. As in [121] we select pages that are popular with our users and we do not consider landing pages. [116] points out another source of uncertainty, namely the fact that mobile pages may greatly differ from their desktop versions. Additionally, in the mobile space, it is likely that other factors, e.g., pages design [141], will play an equally important (if not

greater) role with respect to the protocol pages are served with. As such, we limitedly consider MOS collection for pages in their desktop version.

Measured metrics.
Most works adopt the PLT [120] as single objective performance metric [116, 121,122,123,124,136,137,138,139]. PLT limitations are well established [129, 130], yet only few works include more refined metrics to describe user QoE, such as [137] that consider the SpeedIndex [130]. However, SpeedIndex requires to capture and post-process the video of the page rendering process, incurring in computationally prohibitive costs. More specifically, while the post-processing could be done off-line, the video recording itself slows down the very same rendering process, altering the experiment results (see Sec. 10.2.4). As such, the SpeedIndex is not widely used and its relationship with MOS has never been fully elucidated. Finally, we point out that whereas MOS models for Web traffic do exist [142, 143], their relevance should be reassessed under recent architectures, technologies and designs.

Involving end users in subjective measurements is the best practice to assess actual experience. MOS is the standard in audio and video quality comparison [144], but only recently it has been introduced for WebQoE assessment [145]. To the best of our knowledge, only [140] presents a framework to collect volunteers' feedback on pre-recorded videos of Web-browsing sessions. Side-to-side videos are shown, with the aim of identifying a winner between the experiments. In contrast, we collect volunteers feedback of actual browsing sessions, using the typical 1-5 MOS scale [145]. Both approaches have challenges: Synchronization of videos, correlation between video viewing and browsing experience, ability to slow down/pause video can affect results in [140]. Conversely, the analysis is made complex in our work by volunteers tendency to refraining from using the full scale of score, as well as class imbalance in the gathered dataset, as we shall see.

Overall, given the wide diversity of approaches, it is not surprising to find unexpected or contradicting results, as [116] points out. As we show in this work, using subjective feedback to contrast H2 to H1 can further add complexity to an already quite intricate picture.

9.4.2 CG-NAT Technologies

Given their strategic importance, CG-NATs have been matter of investigation in a large body of studies conducted by both standardization authorities and academia. The Internet Engineering Task Force (IETF) Request For Com-

ments (RFCs) [132, 133, 146] standardize the requirements, implementations and behaviors for CG-NATs. A significant effort has been spent in standardizing mechanisms for NAT traversal, hole punching [134] and Interactive Connectivity Establishment [135].

A remarkable amount of work has been dedicated to the task of identifying NAT deployment in residential networks [147,148,149,150]. Similarly, but in a mobile scenario, [151] presents the results of active measurements to detect the presence of NAT middle boxes deployed in cellular networks.

Similarly to this work, another branch of studies has focused on understanding the impact of CG-NAT on users' Quality of Service (QoS) and application-level experience [152, 153, 154, 155]. Authors of [152] report a comprehensive classification of NATed scenarios and speculate about which impairments each of them could introduce. Authors of [153] presents a set of results obtained in a testbed. Specifically, they analyze the impact of network delays on the TCP connection establishment with and without CG-NAT. In [154] authors evaluate the hole punching technique for NAT traversal, and how this impacts on the communication establishment in P2P applications. More similar in spirit to our work, [155] describes a case study conducted in controlled testbed where multiple CG-NAT configurations are tested to evaluate their impact on several network applications and services. These include Web, video streaming, P2P and gaming, with experiments based on single sessions and not considering actual performance testing. The results presented in [155] show that the presence of CG-NAT has no substantial impact on users' browsing, confirming our observations. Differently, P2P applications like BitTorrent might be severely impaired.

Table 9.1: WebQoE related work at a glance.

Ref.	Experiments scale	Testbed setup		Page catalog	Metrics
		Server	Client (Network)		
[121]	10,000	Squid-proxy + SPDYChrome-proxy	Chrome (3G)	20 popular, no landing	PLT
[123]	n.a. (moderate scale)	Apache, Mono-server, unsharded	Own SPDY client (controlled LAN)	Synthetic + Alexa-200	PLT
[136]	80,000 (500 every hour during 1 week)	Klotski NodeJS proxy on AmazonEC2	Chrome for Android (4G)	50 Landing pages from Alexa-200	Per-user utility
[137]	55,000 (110 configurations per 500 sites)	Apache, multi-server	Chrome HAR capturer (controlled LAN)	Alexa-500	PLT, SpeedIndex
[138]	n.a. (moderate scale)	H2O	Chromium, no TLS (controlled LAN)	Alexa-20	PLT
[124]	n.a. (moderate scale)	Shandian (V8/Blink based)	Chrome vs Shandian (controlled LAN)	Alex-100 (Mobile version)	PLT
[116]	Large scale (crawling, months of experiments)	Internet server (nginx, LiteSpeed)	Chrome HAR capturer (fiber, 3G/4G)	8,492 H2 Websites (200 for 3G/4G)	PLT
[139]	3.4M pages from 40,000 hosts	Akamai logs	Mix of user access (unreported)	Mix of user browser (unreported)	PLT
[140]	43 participants, n.a. experiments	Web-based crowdsourcing , Reply of video captures of H2 vs H1			PLT, User feedback
This work	147 participants, 4,000 experiments	Apache, multi-server, controlled sharding vs unsharding	Chrome, HAR logger (controlled LAN, RTT and packet loss)	1 synthetic + 24 pages ∈Alexa-FR	MOS, PLT, ObjectIndex, ByteIndex

Chapter 10

Assessing WebQoE

In this chapter we aim at assessing the QoE of users when surfing Web pages, i.e., WebQoE. Recent studies have pointed out a direct relationship between users' QoE and the value of business for websites, highlighting the importance of providing means to access the required resources quickly and easily.

We focus on the means to quantify the WebQoE. These can be classified in two main categories, namely objective and subjective metrics. In a nutshell, objective metrics are those that can be assessed in an automatic fashion without requiring users' intervention. On the other hand, subjective metrics explicitly require users' feedback and can be achieved only through data collection campaigns involving real users.

In what follows, we first focus on objective metrics, shedding light on merits and limitations of the current practices and state of the art techniques. We then propose innovative objective metrics for a closer estimation of the actual users' WebQoE. In the second part of this chapter, we focus on subjective metrics. We engineer a testbed for the collection of users' feedback, i.e., MOS grades, to contrast the perceived performance offered by the well-established H1 against the newcomer H2.

10.1 WebQoE Metrics

Tab. 10.1 reports the most prominent metrics to measure WebQoE. In particular, the table groups metrics in four categories:

- **Time-Instant metrics**, which are computed by measuring the time instant a particular event occurs.
- **Time-Integral metrics**, which are computed by integrating over all events of a given type tracked during the progress of a Web page. In

Table 10.1: Metrics to express user perceived quality.

	Metric name	Layer 3	Layer 4	Layer 7	Unit/ Range	Description
Time Instant	TTFB	≈	✓	✓	*sec*	Time at which the first byte of payload is received
	DOM	-	-	✓	*sec*	Time at which the Document Object Model (DOM) is loaded
	TTFP	-	-	✓	*sec*	Time at which the first object is painted
	PLT	-	-	✓	*sec*	Time at which all bytes of payload are received
	ATF [129]	-	-	✓	*sec*	Time at which the content "Above-the-fold" is rendered
Time Integral	ByteIndex	≈	≈	✓	*sec*	Integral of complementary byte-level completion
	ObjectIndex	-	-	✓	*sec*	Integral of complementary object-level completion
	SpeedIndex [130]	-	-	✓	*sec*	Integral of complementary visual progress
Comp. Score	YSlow [156]	-	-	✓	[0,100]	Yahoo's compound score (23 weighted heuristics)
	PageSpeed [157]	-	-	✓	[0,100]	Google's PageSpeed Insight heuristics
	dynaTrace [158]	-	-	✓	[0,100]	dynaTrace's compound score
	MOS	-	-	-	[1,5]	User rating

this category fall the two metrics here proposed, namely the ByteIndex (BI) and the ObjectIndex (OI), which generalize the SpeedIndex (SI) defined by Google.

- **Compound Scores**, weighting altogether several domain-expert heuristics to yield a score in the range [0,100].
- **Mean Opinion Score (MOS)**, computed by averaging users' subjective ratings. MOS can be regarded as a benchmark for the other metrics, but it is admittedly hard to collect MOS points [145].

10.1.1 Time-Instant Metrics

Metrics belonging to this category have the clear advantage of being easily measurable since they only track the realization of a specific event and, as a consequence, are widely used nowadays. Nonetheless, they are arguably simplistic since they disregard the complex chain of events that triggered the measured one. In a nutshell, such metrics compress the whole waterfall chart to a single time instant [131]. Intuitively, two different experiments showing the same time-instant metric could be associated to significantly different user experiences. Despite this, the PLT (also known as onLoad), which measures the time taken to completely load all the objects of a page, is still considered the main key performance indicator in the vast majority of recent scientific works, from both the industrial [125,126] and the academic [121,122,123,124] perspectives.

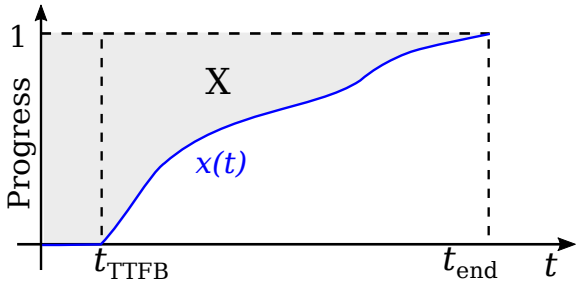


Figure 10.1: Time-integral metrics computation.

Other interesting metrics in this category include the Time to the First Byte (TTFB), i.e., the time instant at which the first byte of payload is received (that expresses the page reactivity), and the DOM event, i.e., the time at which the object-oriented structure of the page is completely downloaded and parsed, after which the rendering can start. Simple tracking of the visual progress is expressed by the Time to the First Paint (TTFP), which measures the time at which the first object is rendered. To further refine the tracking of visual progress, Google proposed the Above-the-fold (ATF) metric, defined as the time at which the content shown in the visible part of the Web page is completely rendered.

It is important to notice that few of these metrics, such as the TTFB, can be measured at the network (L3) or transport (L4) layers, whereas the vast majority, e.g., all those related to rendering, mandates the instrumentation of the browser (L7). Additionally, it is worth to notice that while most of the metrics in this category require few computations (if any), ATF is significantly more complex, as it requires to take screenshots during the rendering process and a post-processing stage of the captured frames.

10.1.2 Time-Integral Metrics

Metrics in this category are characterized by the explicit use of all events in the Web page waterfall [131]. In particular, Google introduced the SpeedIndex (SI) to consider the whole process leading to the visual completion of a Web page and better account for user experience. Here we generalize such a metric and introduce the family of time-integral metrics, defined as follows:

$$X = \int_0^{t_{\text{end}}} (1 - x(t))dt \tag{10.1}$$

where X is the value of the metric, t_{end} is the time the last event is triggered, and $x(t) \in [0, 1]$ is the time evolution of the progress to reach such an event.

Fig. 10.1 illustrates computation of a time-integral metric, where the blue line represents $x(t)$, and the gray-shaded area represents the result of the integral (10.1). Trivially, the smaller the area above the curve $x(t)$, the lower the score X, the better the user experience.

In order to make a concrete example, let us consider the SI [130]. In such a case, $x(t) = \text{painted}(t)/\text{total}$ is the progress of the rendering process, and t_{end} corresponds to the ATF time-instant metric that marks the completion of the rendering. Under this light, the rationale of (10.1) is simple: Not all the sub-events, i.e., the rendering of specific objects, are considered equally important. In particular, (10.1) gives more weight to objects being rendered at the beginning and vanishingly less weight to objects rendered towards the end. In other words, such a metric assigns a lower score to pages (or Web browsers) rendering as much content as possible in the beginning with respect to pages (or browsers) rendering all the objects near $t \approx t_{\text{end}} = \text{ATF}$.

Bounds of Time-Integral Metrics.
It is immediate to notice that the time-integral metric X defined by (10.1) is lower-bounded by t_{TTFB} and upper-bounded by t_{end}. Consider indeed Fig. 10.1, and observe that $x(t)$ is a monotonically increasing function equal to 0 at $t = 0$ and equal to 1 at $t = t_{\text{end}}$. Hence, the worst case time-integral metric is obtained when $x(t) = \mathbb{1}_{t \geq t_{\text{end}}}$ where $\mathbb{1}$ is the indicator function. With such a progress function, all the work is done in correspondence of the event of interest t_{end}. In this case, X is the area of the rectangle of base t_{end} and height 1, i.e., $X = t_{\text{end}}$, which implies $X \leq t_{\text{end}}$. Conversely, the best case is obtained when all the work is done at the beginning.

Notice that in practice, regardless of the considered time-integral metric, no progress whatsoever can be achieved before the first byte of payload (TTFB) is received by the Web browser. Thus, the best case scenario is obtained when $x(t) = \mathbb{1}_{t \geq \text{t}_{\text{TTFB}}}$, which corresponds to the area of the rectangle with base TTFB and height 1, which implies $X \geq \text{t}_{\text{TTFB}}$.

Relationship to Time-Instant Metric.
Extending the above reasoning, it follows that any time-instant metric t_X can be considered as the upper bound of the time-integral metric having $t_{\text{end}} = t_X$ or, in other words, time-instant metrics can be considered as projections of the corresponding time-integral metric. Particularizing this observation to Google's ATF and SI proposals, we have that $\text{t}_{\text{TTFB}} \leq SI = \int_0^{\text{ATF}} (1 - x(t))dt \leq \text{ATF}$, which shows that time-integral metrics allow for a much more fine-grained measure.

Proposed Metrics.
The way the SI metric is actually computed [159] is to take snapshots (by default at a frame rate equal to 10 fps) of a Web browsing session. Such video frames compose a filmstrip which is analyzed in order to infer the visual completion fraction $x(t)$. More specifically, the color histogram of each frame is computed and compared to the histogram of the last frame, which represents the Web page at rendering completion. The use of histograms in place of a per-pixel comparison across frames owes to the fact that during the rendering process new objects might change the position of other objects already rendered. If that is the case, the per-pixel difference would detect the entire section affected by the relocation as incomplete, severely inflating the output value.

However, we show in Sec. 10.2.4 that performing such operations burdens computational resources, significantly inflating the time needed to run the experiment, thus distorting it. To overcome such a limitation, we propose two metrics:

$$\begin{aligned} \text{ByteIndex} &= \int_0^{\text{PLT}} (1 - x_B(t))dt \\ \text{ObjectIndex} &= \int_0^{\text{PLT}} (1 - x_O(t))dt \end{aligned}$$

where $x_B(t)$ and $x_O(t)$ is the percentage of bytes and objects retrieved at time t, respectively. Observe that both metrics require a negligible computational cost, as they can be computed by simply considering the time instants at which bytes/objects are fully downloaded. Finally, both the BI and the OI can be considered as generalizations of the PLT time-instant metric.

The rationale of these metrics is to avoid complex visual captures and leverage the fact that received bytes/objects are directly (e.g., images) or indirectly (e.g., Cascading Style Sheets (CSS)) rendered by the browser. Second, as the SI, these metrics consider all Web page events, with a temporal bias towards earlier events. Lastly, the OI treats all objects equally, whereas the BI introduces a spatial bias as it implicitly states the size of an object to be correlated with its importance for the user.

10.1.3 Compound Scores

Compound scores such as Yahoo's YSlow [156], Google's PageSpeed Insights [157] and dynaTrace [158] encode expert knowledge, usually expressed as a set of heuristics (e.g., 23 in YSlow), combined with heterogeneous weights (e.g., 2% to 30% in YSlow). Such heuristics assess the effectiveness of a Web page design to: (i) Reduce computation (e.g., avoid CSS expressions, alpha

image load, image scaling), (ii) speedup rendering (e.g., limit DOM elements, CSS at the top, JavaScript at the bottom), (iii) reduce data volume (e.g., compress data, minify JavaScript and CSS, use small cookies), and (iv) reduce delay (e.g., reduce DNS lookups, avoid redirections). As such, while relevant to assess the effectiveness of the adopted Web page design and indeed generally used to measure progress/regression of Web pages, these heuristics are unrelated to event timing and hard to map to WebQoE.

10.1.4 Mean Opinion Score

The Mean Opinion Score (MOS) is the ultimate subjective metric for quality assessment in a variety of fields. Originally defined for subjective audio and video quality evaluation [144], new recommendations have recently been proposed for MOS collection in the Web domain [145] as well. The MOS allows the complete measurement of WebQoE from the end user standpoint. However, it is extremely expensive to run comprehensive and statistically significant MOS collection campaigns. As such, automatic approaches have been proposed [129,130] to estimate WebQoE, but their relationship with actual users' experience is yet to be proved, and their computational complexity makes them difficult to use in practice.

10.2 Experiments with Objective Metrics

In this section we characterize *objective* WebQoE metrics, i.e., all the metrics that can be computed by performing measurements at the network level or by instrumenting the browser to record information on the page loading process. In this category fall the time-instant metrics, the time-integral metrics, and the compound scores early introduced. In order to illustrate merits and limitations of these metrics, we use an experimental methodology through which we collect performance indicators in an automated fashion. Our goal is to highlight eventual dependencies or correlations and, more importantly, to verify whether our two proposed solutions, i.e., BI and OI, can be valuable replacements for Google's SI.

10.2.1 Experimental Setup

We consider the top-100 Alexa Web pages (removing regional duplicates) as this is a widely used benchmark both in the industry [160] and academia [136]. As we expect variability across experiments due to load balancing, transient congestion, etc., we repeat the experiments 10 times for each page. For

simplicity, we consider a single browser since differences in rendering and processing engines can play a determinant role. We argue that this would unnecessarily introduce complexity in the analysis and, as such, we consider only Google Chrome being by far the most popular browser and representing more than half of the market share.[1] We use (i) an unmodified Google Chrome (CHR) browser, as well as (ii) WebPagetest (WPT) [159] that we deploy on a local machine to orchestrate experiments. Notice that several metrics, including the SI, are available only via WPT. Conversely, the BI and the OI can be computed via both WPT and CHR.

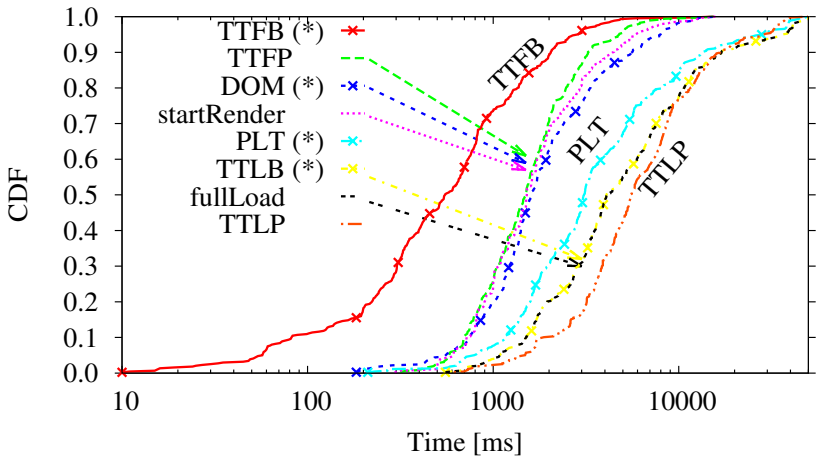
For the time being, we run experiments from a single vantage point located in Paris, which corresponds to the case where CDN nodes are close. Additionally, we consider only the "desktop" version of each website. While the methodology, definition, and metrics would apply to the mobile Web as well, we believe that interactivity of the Web page plays an even greater importance in this latter scenario – which can be quite easy to convince of by considering that mobile Web pages are designed to minimize the visual cluttering and to reduce the time needed to perform a useful action. As such, putting mobile and desktop versions within the same basket would introduce bimodal behaviors in the metrics of interest, which we prefer to avoid.

10.2.2 Results at a Glance

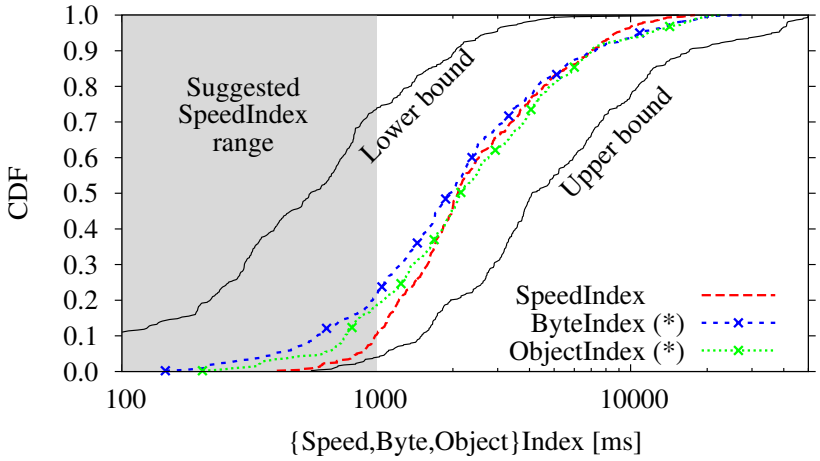
We start by showing in Fig. 10.2 (a) time-instant and (b) time-integral metrics early defined, of which we report their empirical cumulative distribution functions gathered with WPT. In the legend, a star symbol (*) denotes metrics that can be computed via both WPT and CHR.

Considering the *time-instant* first (Fig. 10.2(a)), it can be seen that, as expected [161], events have an order relationship: e.g., no paint (TTFP) can happen before the first byte is received (TTFB), parsing of the DOM is necessary for the rendering process to start, and the reception of the full data (PLT) can happen well before the Time to the Last Paint (TTLP). It can also be seen that the TTFB and TTLP curves constitute the envelope of the process and are separated by over two orders of magnitude, as they pertain to rather different activities. For each metric, it can also be noted a significant variance: The median DOM (PLT) is about 1.5 (3) seconds, while the 90th percentile is above 5 (13) seconds. Finally, it can be observed that some groups of metrics appear closely clustered, e.g., TTFP and startRender; Time to the Last Byte (TTLB) received and fullLoad, hinting for redundancy between the events definition and reporting.

[1] `http://www.w3schools.com/browsers/browsers_stats.asp`



(a) Time-instant metrics.



(b) Time-integral metrics.

Figure 10.2: Characterization of (a) time-instant and (b) time-integral metrics (Alexa top-100) with WebPagetest.

Moving to the *time-integral* next (Fig. 10.2(b)), it can be seen that SI, BI and OI fall between the TTFB and TTLB envelopes. Additionally, these metrics are quite clustered, hinting to the fact that our simpler proposals have intrinsic similarities with the original SI. The dark-shaded region on the left-hand side of the plot highlights the zone of advised SI values for responsive websites. This hints to the fact that WPT slows down the whole rendering process (see Sec. 10.2.4).

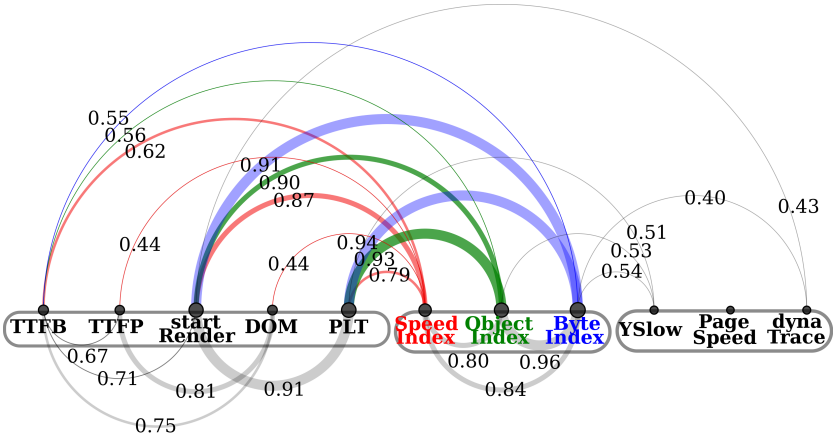


Figure 10.3: Pearson correlation between metric pairs.

A closer look reveals that the BI and the OI *climb faster* than the SI in the short-time frame regime. This is due to the fact that (i) the completion ratio for {Byte, Object}Index increases even before the DOM event, and that (ii) {Byte, Object}Index neglect computational and render time, i.e., they consider bytes/objects useful for the user experience as soon as they are received by the browser. Conversely, {Byte, Object}Index *climb slower* than SI in the tail, as they consider possibly not painted objects (i.e., those that are below-the-fold). While the integral 10.1 from which the {Byte, Object}Index are computed can be fine-tuned to consider other parameters and approximate the SI even more closely (see Sec. 12.2), we believe that the main takeaway is the remarkable proximity and potential interchangeability between the {Byte, Object}Index and the SI, with our proposals being much less expensive in terms of computations.

10.2.3 Relationship among Metrics

To further assess the relationship among metrics, we report in Fig. 10.3 the Pearson correlation matrix between metric pairs, represented as an arc diagram. For completeness, we consider scores that are popular in the industry such as Yahoo's YSlow, Google's PageSpeed Insights, and dynaTrace computed by [160] on the Alexa top-100. In the plot, metrics are arranged into time-instant (left), time-integrals (middle), and compound scores (right). Correlations within the group are reported in gray below the label name, while inter-group ones are reported above. Correlations of the time-integral group are highlighted in red (SI), green (OI) and blue (BI). To improve vi-

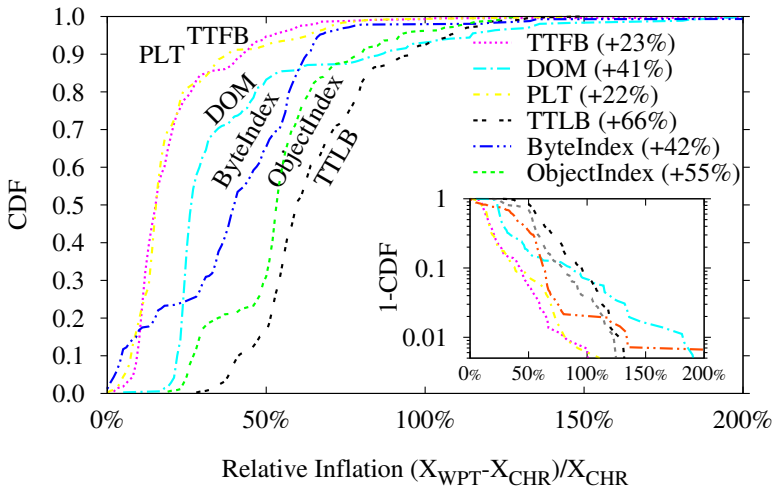


Figure 10.4: Relative inflation of time-instant and time-integral metrics under WebPageTest vs plain Chrome.

sualization, we only report correlations ≥ 0.4, with actual values annotated in the plot. To let the strongest correlation emerge, we quantize line width doubling it every 0.1 steps.

The picture reinforces the soundness of our proposal as it appears that: (i) Our proposed byte-level and object-level replacements exhibit correlation with time-instant metrics, similarly to the SI, and are highly correlated with the SI itself; (ii) YSlow, PageSpeed, and dynaTrace heuristics are poorly correlated among them and with any other WebQoE metric, thus not representing valid alternatives.

10.2.4 Relationship among Experiments

We finally contrast the same metrics gathered via WPT vs CHR. Specifically, WPT computes a larger basket of metrics, notably including those related to rendering (TTFP, ATF, SI, etc.). At the same time, computing such metrics affects the very same experiment. Indeed, as recognized by the community [124], they require cumbersome screen captures that slow-down the rendering process significantly. Indeed, even if the SI can be computed a posteriori, the screen capture itself constitutes a significant CPU bottleneck and remarkably alters the experiment together with the very same user experience the SI tries to capture. It follows that the system bottleneck can possibly move from network capacity to CPU usage, bringing the assessment of the proposed protocols (i.e., H2, QUIC, etc.) in an unexpected/unrealistic

operational point, where many of the potential benefits of the new protocols are hidden by this bottleneck shift.

To quantify the extent of the distortion in WPT vs CHR, we consider the subset of 6 metrics that can be computed in both (i.e., TTFB, DOM, PLT, TTLB, BI and OI) and define as $(X_{\mathrm{WPT}} - X_{\mathrm{CHR}})/X_{\mathrm{CHR}}$ the relative inflation of a generic metric X in the set. The cumulative distribution functions of the relative inflation are depicted in Fig. 10.4, annotating the average inflation for each metric in the label. A zoomed inset shows the complementary CDF, to better assess distortion in the tail. It can be seen that (i) inflation is non-linear, (ii) average inflation ranges from +22% to +66%, (iii) in the worst 10% of the cases the OI is almost doubled (and so is the TTLB). Otherwise stated, distortion in the experiment introduced by computational complexity makes the SI of little practical relevance.

10.3 Experiments with Subjective Metrics

In this section we characterize *subjective* WebQoE metrics, i.e., metrics that require explicit users' feedback, namely the MOS. To achieve such goal, we involve real users volunteering to take part in our collection campaign. Users are asked to watch Web pages being loaded through a dedicated testbed and rate their experience. The testbed allows the control of deployment parameters such as network conditions or the protocol being used to serve content, while the actual browsing is performed through an off-the-shelf Google Chrome browser. This helps in replicating as much as possible the scenario with which users are comfortable and, in turn, maximize MOS significance.

In addition to the MOS, the testbed is instrumented to record objective metrics. This allows us to directly contrast different metrics belonging to the two categories and identify hidden correlations, or, more interestingly, verify whether it is possible to predict the MOS from objective metrics.

10.3.1 Ethical Considerations

When conducting experiments involving people, ethical issues typically arise. In this work, we follow the best practices to minimize eventual issues and discrimination. In organizing the experiments, we follow the "The Menlo Report" guidelines [162], and in particular the best practices for network measurements suggested in [163].

First, we inform the participants on both the goals of the experiments and the methodology used to collect data by giving them an introductory class of 1 hour before each session. Participants are informed about the task

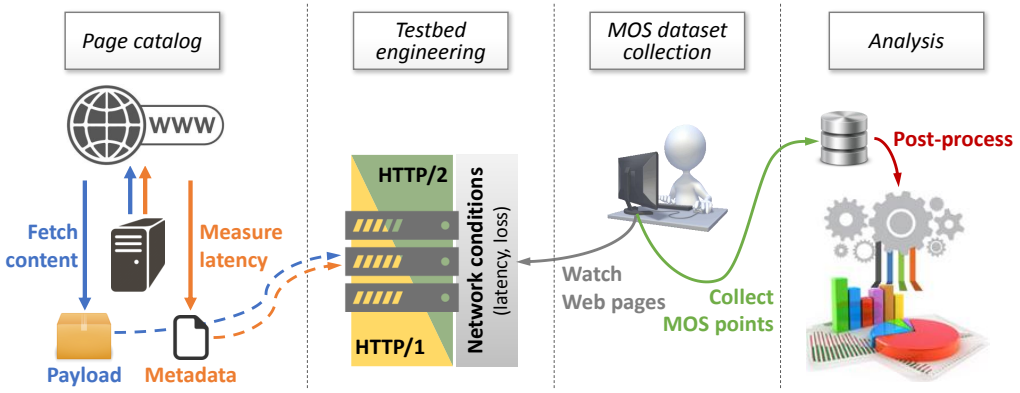


Figure 10.5: Experimental workflow for MOS collection.

to be performed, the duration of their involvement, and the anonymity of data collection. Moreover, participation is voluntary and can be stopped or refused at any time without penalties. The name of a contact person is provided in case of issues or further questions. We do not run any pre-screening or post-screening of participants. No restrictions are applied, e.g., we do not check gender, age, race, citizenship, socio-economical status, etc.

Second, we take all possible precautions to run the data collection sessions so to make participants feel comfortable with the setup and the tasks to be accomplished, e.g., by asking them to run experiments only after a thorough testing of all the process, and offering technical support during the tests.

At last, when running the experiments and collecting results, the anonymity of the participants is guaranteed. Each participant freely chooses where to sit in the lab, so that no association between the computer and the user is possible. A random ID is automatically assigned to each run only to track grades provided by the same user and contrast the perceived performance. It is not possible to associate an ID with a terminal in the lab and, even more, with the actual identity of the participant who provided the grade.

In addition, we take any possible action to further remove eventual traces potentially leading to the identification of the person running the experiments. We delete any timestamp, and the sequence of records is randomized to get rid of any eventual time correlations. Each record contains only metadata about the test and scenario faced, with no indication of the terminal (and user) running the test.

10.3.2 MOS Collection Workflow

To collect users' MOS, we design a system that enables us to download real Web pages from the Internet, deploy them locally to control network conditions, have clients configured to reach local servers, and collect MOS evaluations representative of the user experience in a central repository. The workflow we employ is depicted in Fig. 10.5, and consists of four main steps:

1. **Page catalog (Sec. 10.3.3)** – To build a realistic test, we fetch a number of actual pages and characterize the paths towards servers.
2. **Testbed instantiation (Sec. 10.3.4)** – Pages and paths metadata are shipped to servers in our testbed. We locally host all objects using multiple Apache HTTP servers, controlling network (e.g., RTT, loss), protocol (e.g., H2/H1), and application (e.g., domain sharding) configurations at a fine-grain.
3. **MOS collection (Sec. 10.3.5)** – Volunteers browse pages served by our local infrastructure and provide a score in the range $[1, 5]$. At the same time, we collect all the objective metrics that are not inflating the rendering time, i.e., all objective metrics but ATF and SI (see Sec. 10.2).
4. **Analysis (Sec. 10.4—10.5)** – At a later stage, we apply statistical analysis to the MOS dataset and contrast H2 vs H1 performance.

10.3.3 Page Catalog

We first need to select representative pages to offer users during the MOS tests. We revert to top-100 Alexa Web pages, already used for objective metrics assessment (see Sec. 10.2), but, as our tests take place in Paris, we also consider top-ranked pages in France with which users are likely familiar. We then visit each page using the Google Chrome browser and compile a list of all the URLs of objects being requested by the browser. Some pages are accessed over HTTP, some over HTTPS. We then download and mirror each object on a local server. At the same time of URLs list compilation, we measure the RTT towards each domain providing content.[2] Measurements are performed from our campus network in Paris and are thus representative for a not-congested connection towards content hosted in close-by CDNs.

We manually check each mirrored page from our local servers to both discard incomplete pages (e.g., object failing to download due to dynamic

[2]We use TCP-SYN packets to contact each domain 10 times and compute the average RTT avoiding biased measurements do to potential sporadic network congestion or load-balancing techniques applied at the server side.

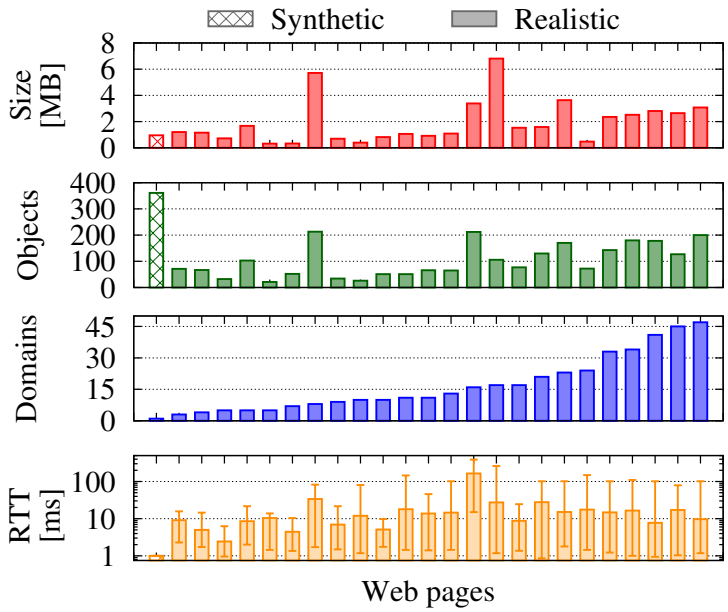


Figure 10.6: Page catalog characteristics.

requests or cookies policies), landing pages [121] (e.g., Facebook login page), etc. We are left with 24 real pages covering a variety of categories, e.g., news, e-commerce, informative websites, leisure etc. At last, we add the toy page `http://www.httpvshttps.com` to the page catalog, for a total of 25 pages. Our page catalog ensures a statistically relevant sampling of popular pages, and it is compatible with the time budget allocated for the experiments and the number of volunteers involved (further information is provided in Sec. 10.3.5). For each considered page, Fig. 10.6 reports its size (top), the number of objects (2nd from top), the number of domains serving such objects (3rd from top), and the average per-domain RTT with respect to contacted domains, with bars reporting the minimum and the maximum RTT (bottom). The figure shows that the benchmark includes very diverse scenarios, from pages hosted on few domains serving a handful of objects, to pages hosted on tens of domains and made of hundreds of objects. Further details concerning the page catalog are available in App. B.1.

10.3.4 Testbed Engineering

This section describes (i) the testbed we engineered and (ii) the collection of MOS grades from volunteers. As for (i), albeit our testbed stems from an independent effort, it is similar to the Mahimahi approach [137]. As for

(ii), to collect MOS grades we adhere to the best practices described in past works [145], as well as to the embrace scientific principle of reproducibility.

Server and Network Configuration.
We design and setup a local testbed where we have full control on both protocols (H1, H2), content placement (sharding), and network conditions (RTT, packet loss). Our testbed is composed of six servers, each equipped with a quad-core Intel processor, 4 GB of memory and two Gigabit network cards connected to the lab LAN. Servers are based on Ubuntu 14.04 with Apache HTTP Server 2.4.18 (note that 2.4.17 is the first release supporting H2 [164]). Apache runs in its default configuration, with H2 and SSL enabled. Content is uniquely served using SSL by installing self-signed certificates.

We run multiple Apache instances as in [137], configured to serve content through virtual hosts, which are both name-based and IP-based. We leverage name-based configuration to distinguish requests directed to different domains being hosted on the same machine, while the IP-based distinction is required to have domains mapped to specific network conditions, as defined below. We next distribute content into each server preserving the original placement of objects into domains and map each domain to a static IP address using the 10.0.0.0/8 range, assigning the addresses to virtual interfaces.

We configure two separate virtual-hosts to serve content using alternatively H1 or H2 so to avoid protocol switching on the client side. The selection of H1/H2 is performed by the client, which directs requests to the IP address of the server implementing the desired protocol. We force all content to be served over TLS so to compare H1 and H2 performance under the same conditions, as browsers do not support H2 over unencrypted connections.

To control network conditions, we use Linux traffic control (`tc`) to set both network latency and packet loss (we do not consider bandwidth limitations in this work). We configure the network to either enforce controlled but artificial network conditions or, for the first time to the best of our knowledge, to truthfully replicate original network conditions (e.g., to replicate path delays measured towards each different domain). In the latter case, since the distribution of RTT among the actual server varies between few ms to hundreds of ms, we quantize the possible RTT into 10 bins (corresponding to the 10 virtual interfaces). To minimize the overall quantization error, we employ the Extremum Seeking [165] algorithm, that we run for each page (in this case, our testbed does not serve all pages at the same time). This results in finding 10 optimal values for the bins, which are in turn the 10 optimal RTTs assigned to the virtual interfaces.

Client Instrumentation.
We make available one PC to each volunteer taking part in our campaign. Each PC runs Linux Mint 17.3, with a set of scripts implementing experiment orchestration. In particular, such scripts (i) setup the local client to reflect the desired scenario, (ii) run Google Chrome v.47.0 to let the volunteer automatically visit a page, (iii) collect the user's score and the objective measurement, (iv) send the results to a central repository.

Each experiment requires several steps to complete. From the users' point of view, the experience starts with a GUI listing all the available websites of the page catalog. Volunteers (i) select a page from the list, (ii) observe Google Chrome that loads the page. At the end, (iii) input the MOS grade, and then (iv) watch again the same page, now served with another protocol. Specifically, the page is loaded using either H1 or H2 in a random fashion at step (ii), and then using H2 or H1 at step (iv). Therefore, users sequentially grade the same page under the same conditions and for both protocols, although they are unaware of the protocol order. This is in contrast with [140] where users are shown screen captures of both protocols side-to-side.

Considering now the implementation point of view, once the volunteer has selected a page, the script (i) configure the system `/etc/hosts` file to direct browser requests to the IP addresses of local servers instead of the public Internet.[3] Two `hosts` files are provided for each Web page to visit, one pointing to IP addresses of H1 servers, the other to H2 servers. Next, the script (ii) starts Google Chrome in full screen mode, disabling the local cache and enabling *in-incognito* mode. This ensures each page is loaded independently on past tests and on eventual cookies. We force Chrome to accept self-signed SSL certificates, and to enable network events logging [166]. The latter are then collected through Chrome remote debugging protocol [167], and dumped into a HTTP ARchive (HAR) file for later stage analysis. Once the user has provided the (iii) MOS grade, (iv) all metadata for that experiment (i.e., HAR log, user's grade, and metadata file with network configuration, timestamp, etc.) are sent to a central repository.

10.3.5 Scenarios and MOS Dataset Collection

We aim at collecting MOS grades in (i) scenarios that are as realistic as possible to provide answers of operational interest, but also in (ii) controlled

[3]Due to the explicit binding between host names and IP addresses in `hosts` file, no DNS resolution takes place. This avoids any bias due to resolution delay and DNS caching, enabling a fair comparison between H1 and H2 performance.

scenarios that the scientific community has already analyzed via objective metrics, to directly contrast these findings.

Given the limited time available with volunteers, the breadth of scenarios that can be explored trades off with the need of collecting a statistically relevant bag of MOS points (from multiple users, for any page in the catalog, for protocols and network settings). Thus, we focus our attention on the following relevant scenarios:

- **Homogeneous network** – Objects are distributed on servers as originally observed, often with sharded domain names. However, RTT and packet loss are artificially forced to be the same for all virtual servers. RTT can be chosen in $\{0, 20, 50, 100\}$ ms, and i.i.d. packet loss rates in $\{0, 1, 2\}\%$. Bandwidth is uncapped. Homogeneous network conditions are tuned as those generally considered in the literature.

- **Heterogeneous network** – Objects are distributed on servers as originally observed, often with sharded domain names. Latency to servers reflects the original RTT measured during the collection process, i.e., on the public Internet. Bandwidth is uncapped, and no loss is introduced. Heterogeneous conditions, instead, introduce realism into the dependency graphs of object download, that may not arise in the case of homogeneous conditions.

- **Unsharded deployment** – All objects are hosted by a single server, on a single domain name and IP. RTT to server can be chosen in $\{0, 20, 50, 100\}$ ms. Bandwidth is uncapped, and no loss is introduced. Unsharded deployment is useful to contrast today's production scenarios (i.e., sharding over multiple domains to circumvent the limitation in the number of TCP connections that browsers can open to any given domain) where H2 is expected to underperform, vs situations that are unrealistic (i.e., all content hosted on a single "unsharded" domain) where H2 benefits are expected to arise [168].

Volunteers are exposed to experiments by randomly choosing one of the possible scenarios reported above (recall that, for each experiment, a user is exposed to both H2 and H1 in a random order). It is important to notice that, in order to avoid biased ratings, no other information but the website name is disclosed to the users. In our campaign, three separate sessions involving independent sets of participants produced a dataset comprising over 4,000 MOS grades, involving 147 participants who sampled a space of 25 pages with 32 different scenarios.

10.4 Results

Performance on the Toy page

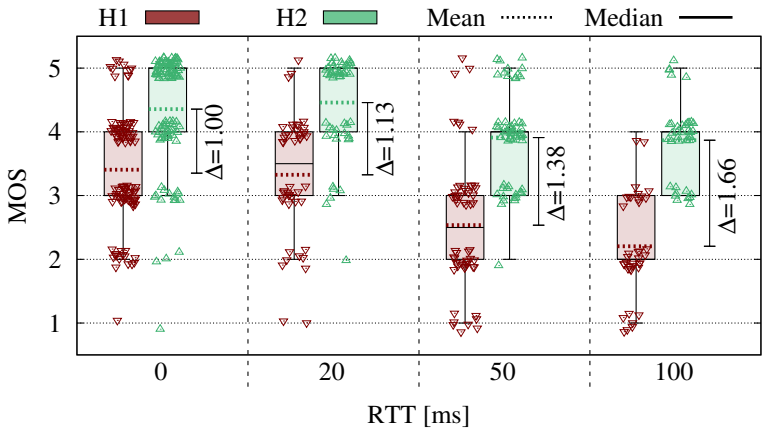
We start the analysis of collected MOS grades focusing on the toy page. For the purpose, we resort to the page `https://www.httpvshttps.com` that is vulgarizing the H2 vs H1 performance difference.[4] We build a similar page comprising an HyperText Markup Language (HTML) document (18 kB) and 360 identical non-cached images representing checkmarks of 20 x 20 pixels (1.7 kB each), for an overall size of 630 kB. All the content needed to render the page is hosted on a single domain, mapped to a single IP address and served by a single machine. Notice that this scenario is particularly adverse to H1 since the browser would open a large number of connections, each of which incurs in TCP and TLS handshaking overhead, and the TCP slow-start. In contrast, H2 should take full advantage of its capabilities and improvements over H1, by pipelining all requests over multiple streams encapsulated in a single TCP connection and by performing HPACK header compression [169]. We expect H2 to reduce the PLT, ultimately providing a sensibly better experience to final users.

We use this toy page to validate the testbed and calibrate MOS grades. On the one hand, we aim at verifying whether expectations on H2 performance are satisfied, confirming the soundness of both methodology and MOS collection procedure. On the other hand, we aim at assessing the extent of MOS gap between H2 and H1 by using this extreme scenario as a litmus paper to define ranges of the performance gap and verifying the existence of the expected logarithmic dependency between delay and MOS reduction [170, 171, 172].

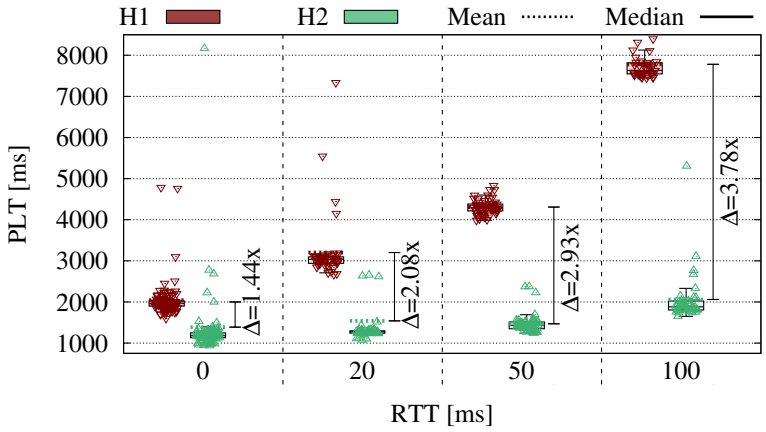
10.4.1 Absolute Performance Assessment

We administered MOS tests to our volunteers considering 4 different network setups, namely with RTT in $\{0, 20, 50, 100\}$ ms, picked at random. We collected overall 487 samples. Fig. 10.7 shows MOS (top plot – Fig. 10.7(a)) and the Page Load Time (bottom plot – Fig. 10.7(b)) for H2 (green) and H1 (red). Each point in the plot corresponds to an experiment, and we add jittering (to x-y axis for MOS and to x axis only for PLT) to enhance the visual presentation.

[4] The URL name and the page header are misleading as, quoting the page, "*Plaintext HTTP/1.1 is compared against encrypted HTTP/2 HTTPS on a non-caching, nginx server with a direct, non-proxied connection.*"



(a) MOS grades vs RTT



(b) PLT vs RTT

Figure 10.7: Boxplots of MOS grades and PLT for homogeneous RTT in the [0,100] ms range. Points represent individual experiments, and coordinates are jittered for visual readability.

Consider MOS first (Fig. 10.7(a)) and notice that the perceived quality of experience consistently decreases with increasing RTT. This holds for both H1 and H2, with H2 providing a better experience at both low and high latencies. Notice also that the difference between the average of H1 and H2 MOS grades is always of at least 1 point, and increases along with the RTT, peaking at 1.66 points for RTT= 100 ms.

Consider now PLT in Fig. 10.7(b). It emerges that loading time increases along with an increased RTT. H1 is significantly more penalized than H2, with PLT peaking at more than 8 s for RTT= 100 ms, while H2 keeps PLT

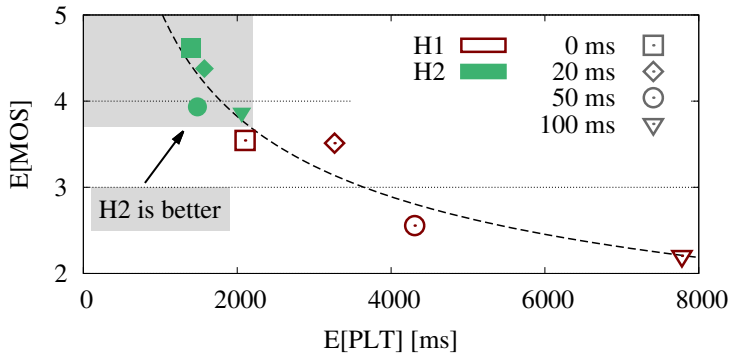


Figure 10.8: Sub-linear dependency between PLT increase and MOS reduction, synthetic page.

below 2 s in all scenarios. In a nutshell, H2 significantly outperforms H1 PLT, meeting the original design goal of "a 50% reduction in page load time" [20].

We further verify the existence of a sub-linear dependency between the subjective response to an objective impulse [172]. In particular, the impulse is here represented by the additional RTT, that translates into a longer PLT, while the response is the MOS degradation.

Fig. 10.8 reinterprets Fig. 10.7 as a scatter plot, where each point represents the average $(\mathbb{E}[\text{PLT}], \mathbb{E}[\text{MOS}])$ pair, computed over all samples for a given RTT setting. For illustration purposes[5], the figure reports the trend line. The expected sub-linear dependency clearly emerges.

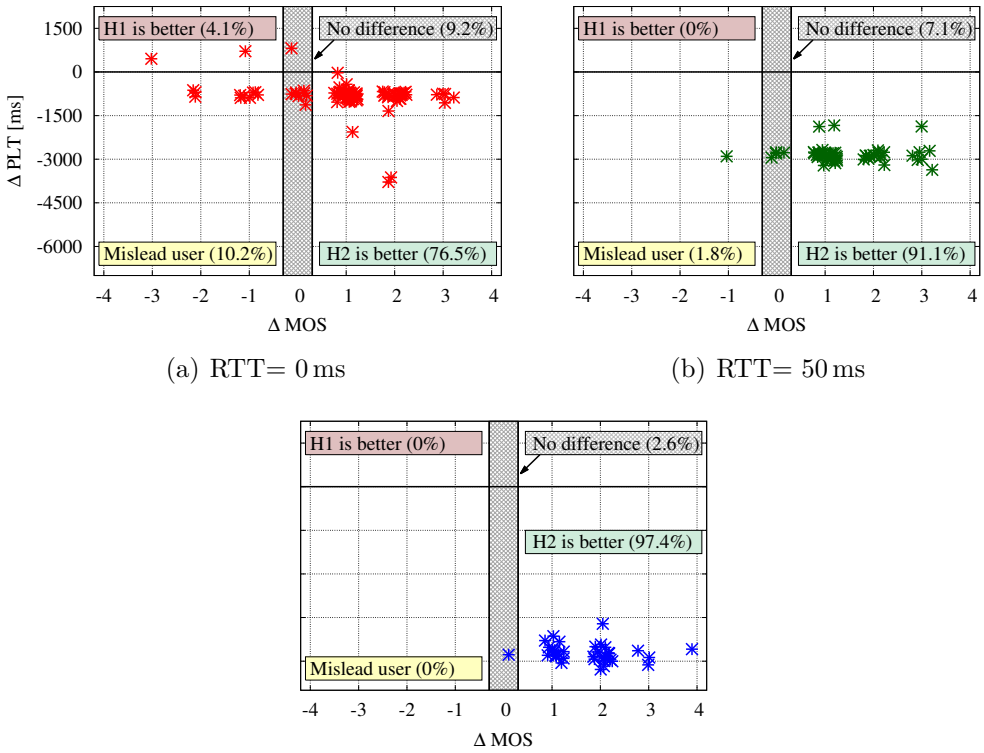
A final remark is that excellent MOS (i.e., 4 or higher) is bestow only to pages with a loading time lower than 1.25 s (i.e., H2 with low RTT). This is in line with [173], which classifies pages as *reactive* if they render the above-the-fold content in less than 1 s.

10.4.2 Relative Performance Assessment

Given each user tested always both H1 and H2 setup, we now quantify the perceived improvement (or degradation) in the loading process on a per-user basis. Let us denote the difference in the score given by user u browsing the page w under settings s as follows:

$$\Delta\text{MOS}_{u,w,s} = \text{MOS}_{\text{H2},u,w,s} - \text{MOS}_{\text{H1},u,w,s}$$

[5]Notice that our aim is not to provide an exact fitting – which would be of limited interest as we are considering a simple artificial page in this case.



(a) RTT= 0 ms (b) RTT= 50 ms

(c) RTT= 100 ms

Figure 10.9: Scatter plot of ΔMOS vs ΔPLT. Points are jittered for the ease of visualization. Users hardly appreciate (or are even mislead by) differences in the low RTT case. Improvements are instead solid for increased RTTs (50 ms or higher), for which ΔPLT becomes larger.

A positive value of ΔMOS implies a better quality of experience with H2. Similarly, we denote with:

$$\Delta\text{PLT} = \text{PLT}_{\text{H2},u,w,s} - \text{PLT}_{\text{H1},u,w,s}$$

the page completion time difference, where a negative ΔPLT value implies better H2 performance.

Fig. 10.9 depicts the ΔPLT (y-axis) and the ΔMOS (x-axis, adding jitter noise for visual presentation) as a scatter plot, where each point represents the realization of an H2+H1 experiment pair from a user. We provide results for RTTs equivalent to 0, 50 and 100 ms.

Each plot in Fig. 10.9 can be partitioned in five regions:

(i) South-East. H2 is faster ($\Delta\text{PLT} < 0$) and users prefer H2 over H1

(ΔMOS > 0), i.e., H2 is better according to both subjective and objective evaluations. We observe most experiments to fall into this region (over 75%), especially at high RTT (over 97% with RTT$= 100$ ms).

(ii) North-West. H1 is faster (ΔPLT > 0) and users score H1 better than H2 (ΔMOS < 0), i.e., H1 is better according to both subjective and objective evaluation. We observe few points (4%) falling into this region, and only in the case of RTT$= 0$ ms.

(iii) South-West. H2 is faster (ΔPLT > 0), but users prefer H1 over H2 (ΔMOS < 0). This is a region where subjective and objective evaluation appears not to be consistent, suggesting that users are mapping the difference in ΔPLT to the difference in ΔMOS in a counter-intuitive way. Surprisingly, we find a non-negligible fraction (10%) of experiments falling in this region, though only for the very low RTT setting.

(iv) North-East. H1 is faster (ΔPLT > 0) but users prefer H2 over H1 (ΔMOS > 0). We observe no points falling in this region.

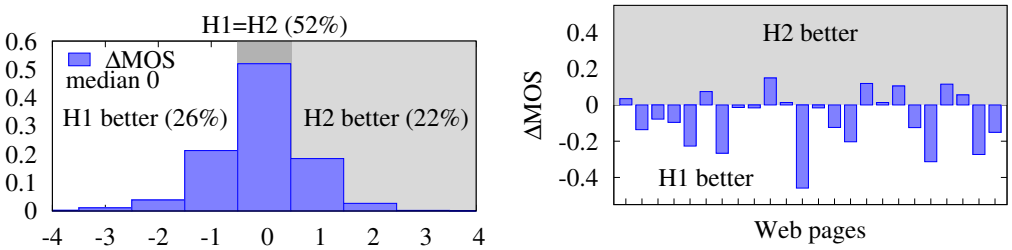
(v) Vertical Strip. Finally, we highlight a gray-shaded zone where no differences in MOS are reported by users (ΔMOS $= 0$), irrespectively of the ΔPLT gap. In this region falls a non-negligible fraction of experiments, decreasing as RTT increases: From 9% at RTT$= 0$ ms to slightly less than 3% at RTT$= 100$ ms.

Two main considerations hold. First, with respect to Fig. 10.8 (showing a simpler and expected dependency between mean $\mathbb{E}$[PLT] and $\mathbb{E}$[MOS]), Fig. 10.9 depicts a more complex relationship. We observe that similar differences in the ΔPLT space (notice the small variance for a given RTT settings) correspond to a fairly large range in the ΔMOS space. Second, while at first sight one could attribute unexpected results to unreliable users, notice that, for increased RTT, unexpected results vanishes and H2 is consistently indicated as a winner, although the dispersion of ΔMOS grades remains quite large. These observations suggest that users provide honest answers, albeit the subjective score of individuals varies a lot. We point out that these results are in line with related studies such as [136].

10.5 Results

Performance on real-life Web Pages

We now focus on real-world pages and quantify differences between H2 and H1, from both subjective MOS grades, as well as from objective metrics. With respect to the previous section, we now include a larger basket of met-



(a) Empirical probability mass function.

(b) Per-page MOS difference.

Figure 10.10: H1 vs H2 MOS grades (ΔMOS) for all 4,000 tests.

rics (introduced in Sec. 10.1) and pages (Sec. 10.3.3). Our aim is to assess if and to what extent differences in user experience shown earlier with the toy page (Sec. 10.4) hold for actual and more complex pages as well.

10.5.1 Subjective MOS Differences

We start by assessing the H2 vs H1 difference of MOS grades per-user for each page in the catalog. Fig. 10.10 shows ΔMOS over all tests, detailing both the empirical probability mass function (Fig. 10.10(a)) and the per-page MOS difference (Fig. 10.10(b)). The figure is annotated with statistics (e.g., median) and visual references (e.g., light-gray area for H2 better than H1 – $\Delta\text{MOS} > 0$; dark-gray area where a difference in quality is not distinguishable – $\Delta\text{MOS} = 0$).

Some insightful observations can be drawn from the plots. The distribution in Fig. 10.10(a) is (i) monomodal with zero mean and median, (ii) bell-shaped, but (iii) slightly skewed. In other words, (i) in 50% of cases, users equally score H2 and H1, (ii) cases where either H2 or H1 has higher score are roughly balanced, although (iii) there is a slight yet noticeable bias towards negative ΔMOS, where H1 MOS is higher than H2 one. That is, contrary to the previous results, now the difference between H2 and H1 is much less appreciable and much less consistent.

This reinforces the need to perform experiments on real-world pages, as opposite to artificial pages that inflate MOS differences. Results are only partially surprising. First, pages widely differ (see Fig. 10.6) and ΔMOS varies according to the page being considered, as shown in Fig. 10.10(b) (the order of web pages is consistent Fig. 10.6). Second, users have a different way to value improvement, causing them to report the same score under both protocols, which contributes to $\Delta\text{MOS} = 0$. Third, pages in our catalog are likely optimized for H1. Fourth, consider that whereas the H1 software

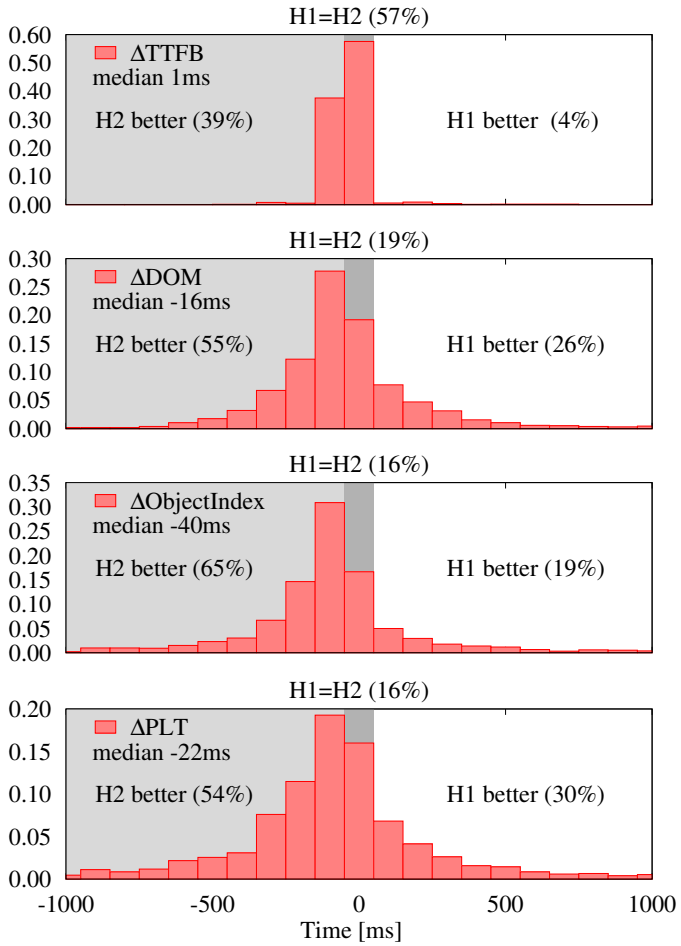


Figure 10.11: Empirical probability mass function of ΔOBJ, for various objective metrics.

has undergone decades of testing and optimization, H2 is a relatively new protocol that has not enjoyed a comparable amount of work on its codebase.

10.5.2 Objective Metric Differences

We next study the H2 vs H1 difference via objective metrics (OBJ in short). As before, we quantify the difference in the delay that a user experienced in accessing the same page over the two different protocols with $\Delta\text{OBJ} = \text{OBJ}_{\text{H2},u,w,s} - \text{OBJ}_{\text{H1},u,w,s}$, where OBJ is any objective metric such as the Time to the First Byte (TTFB), the Document Object Model (DOM), and Page Load Time (PLT). We additionally consider the ObjectIndex, a replacement

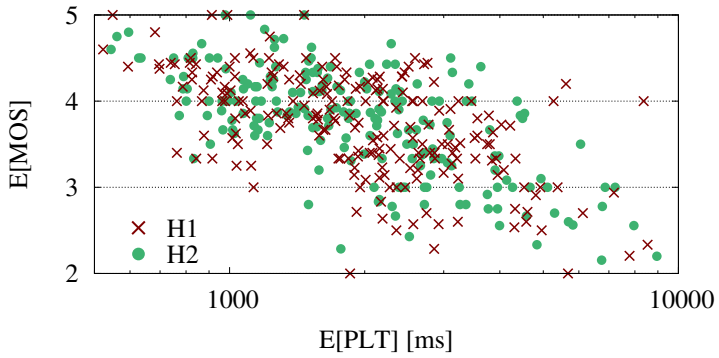


Figure 10.12: Scatter plot of $\mathbb{E}$[MOS] vs $\mathbb{E}$[PLT] (semilog-x) for each page and scenario.

metric for the SpeedIndex [130] that does not face the same computational complexity and has been shown to be strongly correlated with the latter on the Alexa top-100 websites (see Sec. 10.1.2 – Proposed Metrics). Notice that these metrics respect an order relationship, i.e., for any given dependency graph of objects in a page, TTFB $\leq$ DOM $\leq$ ObjectIndex $\leq$ PLT.

Fig. 10.11 depicts the distribution of ΔOBJ over the very same experiments of ΔMOS in Fig. 10.10(a). ΔOBJ density is discretized in bins of 100 ms. As before, the figure is annotated with statistics (e.g., median) and references (e.g, notice that H2 is better than H1 is represented by the negative semi-plane in this case). A couple of observations are worth highlighting. First, notice that all OBJ exhibit an empirical probability mass function that is similar to that of the ΔMOS grades (i.e., roughly symmetric, peak close to zero, very low median). Second, notice that the dispersion increases with the increase of the metric support (which follows from the order relationship). Third, and most important, notice that OBJ attribute a (slight) advantage to H2, unlike in the MOS case.

We finally show in Fig. 10.12 the relationship between the MOS and OBJ metrics over the whole dataset, arranged as a non-jittered scatter plot in which each point $(\mathbb{E}[\text{MOS}], \mathbb{E}[\text{PLT}])$ represents the mean MOS and PLT computed over all users having seen the same page in the same scenario. In this case, whereas a sub-linear dependency between MOS and OBJ can still be observed for both protocols (notice the logarithmic x-axis), H2 and H1 grades are no longer clearly separated as early shown in Fig. 10.8 for the toy page. Of course, while this is not surprising in light of the large number of

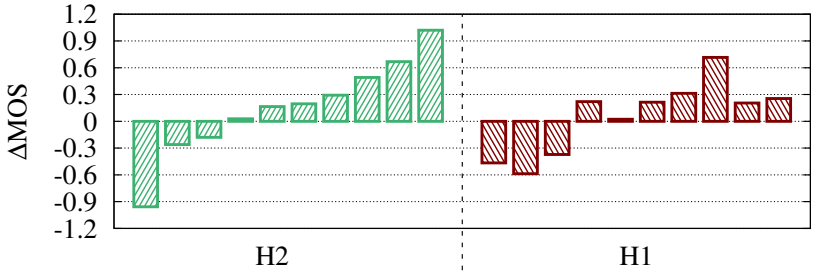


Figure 10.13: ΔMOS for 10 pages where we contrast unsharded vs sharded versions.

ΔMOS $= 0$ points shown in the previous section, it also anticipates a more complex relationship between OBJ and MOS than early illustrated.

10.5.3 Impact of Domain Sharding

We now consider sharding [168], i.e., distributing page content over multiple domains to exploit server parallelism. This practice helps in overcoming the limitation on the maximum number of connections a browser can establish towards the same domain. Given H2 benefits of using a single connection to a single domain [116, 124, 136], one would expect that unsharding helps in taking advantage of H2 pipelining features.

In our evaluation, we consider 10 of the 25 pages of the catalog and modify them so to have all the content hosted on a single domain (i.e., unsharding the content). We then contrast MOS grades to assess the impact of (un)sharding for H2 and H1 independently. Fig. 10.13 shows the per-page difference between the average MOS for the unsharded and for the sharded content. In formulas, $\Delta\text{MOS} = \mathbb{E}[MOS|_{\text{unsharded}}] - \mathbb{E}[MOS|_{\text{sharded}}]$. Pages are sorted increasingly according to ΔMOS for H2.

It is straightforward to notice that the impact of sharding is page-dependent. There are pages for which the user experience improves when they are served through the unsharded deployment ($\Delta\text{MOS} > 0$), as well as pages suffering from usharding ($\Delta\text{MOS} < 0$). 7 pages out of 10 show an improvement in MOS when unsharded, even though the difference in perceived quality greatly changes, from a minimum of 0.028 to a maximum of 1.020 ΔMOS points. H2 appears to benefit more of unsharding, but 3 pages get a sensibly reduced MOS. H1 is less impacted, peaking at a difference of "only" 0.716 ΔMOS points.

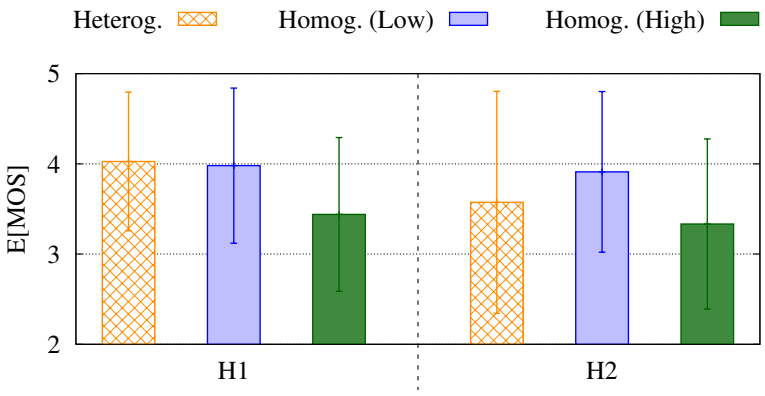


Figure 10.14: Testbed realism: $\mathbb{E}$[MOS] for heterogeneous vs homogeneous RTT settings.

10.5.4 Impact of Latency Diversity

Finally, we come back to the trade-off between realism of experiments conducted in the wild Internet, but lacking control [116, 121, 136, 139], and full control achievable in a controlled testbed, lacking realism [123, 137, 138]. Specifically, we contrast scenarios with *homogeneous RTT* (usually considered in the literature [123, 124, 137, 138]) against *heterogeneous RTT* to each domain (fine-grained control that, to the best of our knowledge, we are first to consider in a testbed). We point out that while Mahimahi [137] could in principle allow the setup of multiple DelayShells, one for each domain, this is not done in the evaluation. Moreover, the collection of path characteristics, i.e., RTT, and the per-page configuration and deployment are quite a tedious tasks that our testbed fully automates.

Clearly, homogeneous conditions make sense in case of proxy-based proposals [121, 124, 136] and is typically justified in testbed studies with the assumption that sharding practice will ultimately be abandoned as countering H2 benefits – which might not alway be true, as seen in the previous section. At the same time, sharding is a standard practice, and page redesign would happen conditionally on un-sharding benefits being proved and consistent. Additionally, complete un-sharding is unlikely as pages aggregate many contributions (e.g., twitter feeds, advertisement, etc.) coming from multiple producers. As such, it is important to evaluate H2 performance also in controlled conditions that are as close as possible to the real ones.

For this experiment, we select a coherent subset of 3 pages sampled by 95 users for a total of 362 experiments. Average MOS scores are reported in Fig. 10.14. It can be seen that different testbed settings can lead to *opposite*

biases for H1 vs H2. For instance, in the case of H1, high-RTT homogeneous scenarios (where all the domains are over 50 ms away) lead to a loss of about 0.5 MOS points with respect to both heterogeneous (where only few domains are high RTT) and low-RTT homogeneous scenarios. Conversely, in the case of H2, low-RTT homogeneous scenarios lead to an optimistic estimation of MOS grades (less than 0.5 points in our dataset), whereas heterogeneous or high-RTT homogeneous scenarios have closer performance.

As we (and [139]) have noticed, macroscopic pages characteristics are not telling as for user MOS. We suspect this performance gap to be rooted in page dependency graph [124, 174]. Moreover, homogeneous latencies may hide intricate interactions in the dependencies, that may instead arise under heterogeneous conditions.

10.6 Conclusions

In this chapter, we tackled the problem of measuring the quality of experience for Web users. We provided a comprehensive view of metrics used today to assess WebQoE, highlighting their merits and limitations. In addition, we generalized Google's SpeedIndex and introduced two new metrics, namely ByteIndex and ObjectIndex, that retain the same representativeness of users' QoE but have a negligible computational complexity.

We then contrasted the performance of HTTP/1.* against the newly introduced HTTP/2 considering the above metrics, as well as collecting over 4,000 MOS grades provided by real users. For the purpose, we engineered a framework which allowed us to replicate real Web pages in a controlled environment where we can configure network parameters (e.g., latency, loss), the protocol being used (i.e., H1, H2) and domain sharding.

Results do not allow the election of a winner. While H2 sensibly reduces the PLT on a toy page, ultimately improving the Quality of Experience, it is not as effective when serving real-world Web pages. Objective metrics (e.g., DOM, PLT, etc.) show a performance improvement to the advantage of H2 in more than 50% of cases, but they are in disagreement with users' MOS, which is higher for H1 instead.

Clearly, given the limited span of the data at our availability (i.e., number of Web pages considered, skew in user population, combinatorial dispersion of network settings, etc.), we cannot pretend our dataset to be representative of the average browsing experience of an Internet user. As such, collecting and sharing additional MOS datasets can be helpful in building a more comprehensive assessment of H2 performance from the user standpoint.

Chapter 11

Impact of Carrier Grade NAT

In this chapter we focus on the impact that middle boxes deployed by ISPs in their network might have on users' traffic. Specifically, we direct out attention to the understanding of Carrier Grade NAT (CG-NAT) impact on Web traffic. Network Address Translation (NAT) technologies have been deployed by ISPs over the last years to alleviate IPv4 exiguity and reduce the expenses of their high rental cost. CG-NAT is a well-known solution to mask an entire subnetwork behind a limited amount of public IP addresses. It is however not clear if CG-NAT affects users' traffic performance.

We leverage passive traffic measurements collected in an ISP network where CG-NAT devices are deployed. By means of statistical techniques, we contrast several performance metrics considering customers being offered public or private IP addresses. In particular, we focus on metrics representative of Web traffic performance, gaging the impact of CG-NAT presence on users' Web browsing experience.

In the following, we describe the workflow designed to quantify the impact of CG-NAT. Starting from network traffic traces, we describe how to build empirical distribution functions and the statistical tools needed to contrast them. We then apply such techniques to the specific use case of CG-NAT, empirical distributions produced by users with a private or public IP address.

11.1 Methodology Overview

We design a workflow aiming at automating the collection of network traffic and its following processing. In order to extract meaningful indications from the raw network traffic, the definition of representative analytics is of paramount importance. We look for the identification of high-level perfor-

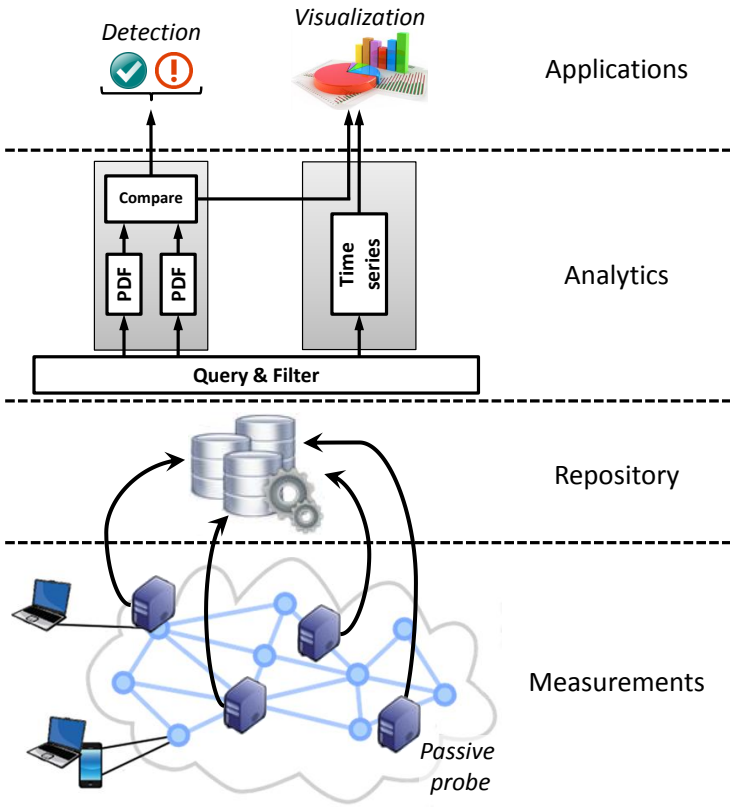


Figure 11.1: Illustration of the measurements framework.

mance metrics enabling the detection of differences in network traffic measurements belonging to different populations of users, periods of time, etc.

Fig. 11.1 illustrates our workflow. Several layers are represented. From the bottom, the measurements layer consists of passive Tstat probes installed in an operational network. Probes are responsible for extracting higher-level statistics from raw packets. Such statistics play the role of KPIs in our analysis and are better detailed in Sec. 11.3.1. The traffic summaries so produced are asynchronously moved to a central repository where data is gathered from multiple sources.

A "Query & Filter" engine allows the access to the repository for the extraction of specific measurement samples (e.g., select RTT for TCP connections where application layer protocol is HTTP, server name matches **.google.com*, client IP address is private, etc.).

Empirical Probability Distribution Functions (PDFs) and CDFs are then estimated from the samples of interest using a simple module that, given the

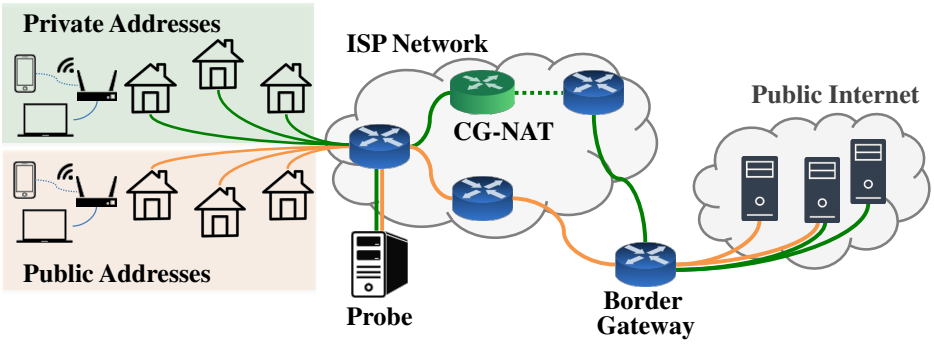


Figure 11.2: Monitoring scenario for Web traffic.

size of bins and support range, computes the frequency of samples falling in each bin, i.e., the probability p_i that the sample takes values in the i-th bin.

Finally, PDFs are compared through statistical techniques so to quantify the distance between empirical distributions built from traffic samples belonging to different populations of users, i.e., contrasting the performance differences between users with a private IP address and users with a public one. The obtained results are targeted to detecting eventual performance issues, while a visualization engine is responsible for reporting the high-level output of the performed analysis.

11.1.1 Measurement Layer

To characterize the implications of CG-NAT, we rely on passive measurements obtained by instrumenting a monitoring probe in the operational network of an European country-wide ISP. Each customer device accesses the Internet via an ADSL home router. The ISP assigns either a public or private IP address to each home router according to the customer's subscription type. Fig. 11.2 depicts our monitoring scenario. We install a passive probe, namely Tstat, (see Sec. 1.2.1) at one PoP of the ISP to monitor the traffic generated by home routers having either a public or a private IP address. Traffic directed to the Internet and coming from home routers with a public IP address (public home routers) is routed directly to the final destination (orange lines), while traffic from home routers with a private IP address (private home routers) has to cross the CG-NAT device first (green lines).

The CG-NAT used by the monitored ISP is based on the NAT444 standard [175], which relies on sessions to translate the private, edge-facing IP address of a home router into a public, Internet-facing one. When the CG-NAT receives the first packet from a private home router, it starts a new

Table 11.1: Traffic produced by home routers with private/public IP.

	Private	Public
TCP flows	990 M	767 M
UDP flows	2,676 M	1,941 M
Failed-TCP flows	301 M	347 M
Traffic Volume	168 TB	105 TB

session, temporarily mapping the private address to the first available public address in a pool. It then converts the address of all subsequent packets according to the same mapping. [1] After a given inactivity time during which no packets are observed, the session expires and the public address is put back in the pool of available addresses.

11.1.2 Dataset Description

We leverage a dataset collected during the month of October 2014. It consists of TCP, UDP and failed-TCP logs carrying 1,757 M, 4,617 M and 648 M records respectively, for a total of more than 273 TB of network traffic. As we target the performance assessment for Web browsing, we specifically focus on flows carrying either HTTP or HTTPS transactions. Overall, we process more than 400 M TCP flows containing 688 M HTTP requests, ensuring that the dataset at our availability is large enough to avoid biases due to population size or imbalance, and performance indicators are not affected by quantization artifacts.

We split the dataset in two subsets according to the IP address type, i.e., private or public, of the customer's home router. Tab. 11.1 provides statistics for the two populations. In total, we find more than 20,000 active home routers. Out of these 60% (40%) are assigned a private (public) IP address.[2]

[1] The amount of public addresses available at the NAT is smaller than the number of customers provided with a private IP address. Consequently, the pool size of public addresses must be carefully set to minimize allocation costs, while guaranteeing satisfactory connectivity. See App. C.1 for a thorough discussion.

[2] The home router IP address can be considered as an identifier of the household. It may hide several devices connected to the Internet.

11.2 Quantify Statistical Differences

In this part, we focus on techniques to compare measurements referring to different populations of users, i.e., that help us to quickly pinpoint eventual performance differences between customers provided with a public or private IP address. In what follows, we present an overall view and the necessary elements to understand the potential applications of such analytics.

11.2.1 Comparison and Quantization Functions

The comparison of PDFs poses significant challenges the analyst has to cope with. It is of primary importance to select a proper statistical method to compactly quantify the difference between two distributions. The output can be "soft", i.e., a real value in a continuous range, or "hard", i.e., a categorical output from a (small) set of possible values. For instance, in the first case, comparison tells the analyst how different the statistics are, while in the second case it just tells if they differ or not.

In formal terms, the comparison function has the form $F(p,q) : (\mathbb{R}^2, \mathbb{R}^2) \rightarrow \mathbb{R}$, while the quantization function can be defined as $Q(F(p,q)) : \mathbb{R} \rightarrow \mathbb{N}$, where $p = p(x)$ and $q = q(x)$ are two empirical distributions under analysis.

Without loss of generality, we define a simple quantization function that considers three possible levels, correlating with a no difference state (0), a definitively different state (2), and a possibly different state (1) requiring further, though likely not urgent, investigation. Such quantization function can be written as:

$$Q(F(p,q)) = \begin{cases} 0 & \text{if } F(p,q) < Q^- \\ 1 & \text{if } Q^- \leq F(p,q) < Q^+ \\ 2 & \text{if } F(p,q) \geq Q^+ \end{cases}$$

with states discriminated by the lower Q^- and upper Q^+ thresholds. Selecting these thresholds requires careful attention since (i) there is a dependency between the thresholds Q^-, Q^+ values and the comparison function $F(p,q)$; (ii) the value of $F(p,q)$ can be noisy when distributions p and q are computed over small population samples; (iii) the value of $F(p,q)$ can be affected by class imbalance when population samples of distributions p and q are of different orders of magnitude; and (iv) the value of $F(p,q)$ can be affected by the measurement process (e.g., binning strategy, number of bins, etc.).

11.2.2 Jensen-Shannon Divergence

We make illustrative examples of one Statistical Distance Measure (SDM) on controlled distributions p, q. As representative SDM in this class, we take the Jensen-Shannon Divergence (JS), which is defined as:

$$JS_{div} = \sum_i \Big\{ \frac{1}{2} p_i \ln\Big(\frac{p_i}{\frac{1}{2}p_i + \frac{1}{2}q_i}\Big) + \frac{1}{2} q_i \ln\Big(\frac{q_i}{\frac{1}{2}q_i + \frac{1}{2}p_i}\Big) \Big\}$$

where p_i and q_i are the empirical probabilities of samples taking values in the i-th bin. JS is a popular statistical measure based on the Kullback-Leibler divergence, over which it brings some notable improvements, adding symmetry, i.e., $JS_{div}(p,q) = JS_{div}(q,p)$, and bounded support, i.e., $JS_{div} \in [0, \ln(2)]$. JS is equal to 0 if $p == q$, while it saturates to $\ln(2)$ for two completely disjoint distributions.

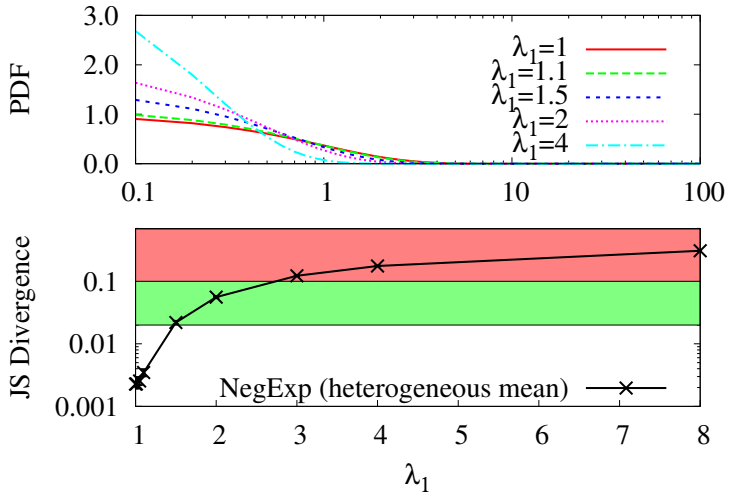
We focus on understanding how the JS varies when comparing two synthetic PDFs, with the aim of defining Q^-, Q^+ thresholds to separate the areas into three states in a generic case. To this extent, we consider (i) negative exponential distributions with different mean; and (ii) Gaussian distributions with different mean and/or different standard deviation. These PDFs are representative of diverse properties that may appear in network data; e.g., packet inter-arrival times and packet size in VoIP calls can be approximated with Gaussian distributions [176]; requests generated by user activities are well approximated with Poisson processes and have as such negative exponential inter-arrival times [177, 178].

For the sake of the example, we now quantify differences between controlled PDFs that are representative of fictionary, yet plausible, processes. Such analysis is useful both to visually tie the JS behavior to some well-known distributions and to identify quantization thresholds that discriminate among significant (2), noticeable (1) and negligible (0) differences between PDFs. The purpose is to illustrate the methodology we followed in setting the quantization function $Q(f(p,q))$.

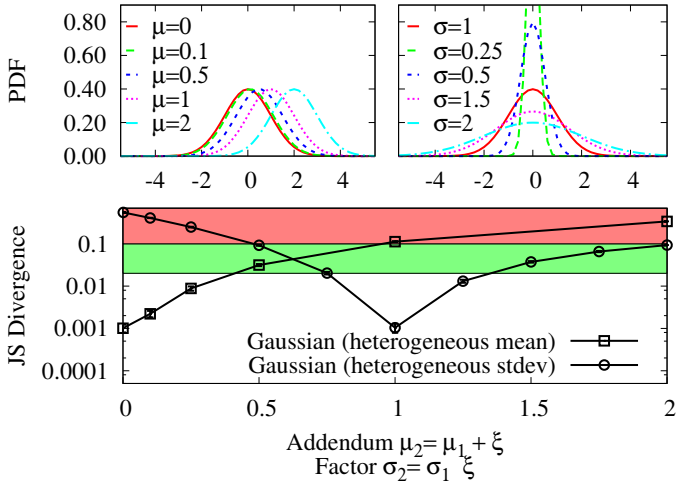
We consider a dataset where samples are extracted from synthetic p, q. Let consider first the comparison of p, q which are both negative exponential distributions:

$$NegExp(x, \lambda) = \lambda e^{-\lambda x} \text{ for } x \geq 0;\ 0 \text{ otherwise}$$

We consider $p = NegExp(x, \lambda_0)$ as a reference, and choose λ_0=1, while $q = NegExp(x, \lambda_1)$ is instead shaped according to a distribution of parameter λ_1, with $\lambda_1 \in [1, 8]$ in our experiments. From both distributions, we extract 10^6 samples, obtain the empirical PDFs using 1000 bins in a $[0, 100)$ support. This leads to bins of size $\Delta b = 100/1000 = 0.1$. For each bin i, we estimate



(a) Negative exponential distribution



(b) Gaussian distribution

Figure 11.3: Illustrative examples of Jensen-Shannon divergence computed on (a) Negative exponential distributions with heterogeneous mean rates λ_1 vs reference mean $\lambda = 1$; (b) Gaussian distributions with heterogeneous mean $N(\mu, 1)$ or standard deviation $N(0, \sigma)$ vs reference distribution $N(0, 1)$.

p_i and q_i as the ratio between the number of samples falling in the i-th bin, i.e., $[i\Delta b, (i+1)\Delta b)$, and the total number of samples.

Negative exponential PDFs p and q are depicted in the top portion of Fig. 11.3(a), whereas the bottom plot reports the JS versus λ_1. We select

thresholds $Q^- = 2/100$ and $Q^+ = 1/10$ so that a clearly visible changes in the distribution space (top) are visible in the JS space (bottom) as well. Intuitively, when $JS_{div} \in [Q^+, \ln(2)]$, the difference between the two PDFs is significant (red area). When $JS_{div} \in [Q^-, Q^+)$ the difference is noticeable (green area), and negligible if $JS_{div} \in [0, Q^-)$ (white area).

We repeat the experiment considering Gaussian distributions, i.e.,

$$N(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

As before, we generate a reference sample p corresponding to $(\mu, \sigma) = (0, 1)$, and samples q with different (μ, σ) parameters. The upper-left plot of Fig. 11.3(b) shows PDFs of q with parameters $\mu \in \{0, 0.1, 0.5, 1, 2\}$ and σ=1, while the upper-right plot shows PDFs when μ=0 and $\sigma \in \{0.25, 0.5, 1, 1.5, 2\}$. Fig. 11.3(b) lower plot reports the JS values when comparing the above-mentioned distributions against p.

The previous threshold selection proves to be effective also in the case of Gaussian distributions: Visible differences in the upper plots of Fig. 11.3(b) appear to be separated by the $Q^- = 2/100$ and $Q^+ = 1/10$ thresholds.

In real cases, the sensitivity of a domain expert can be used to set thresholds. In general, any threshold choice results arbitrary, which applies to any SDM of choice (and possibly being even more complicated for those SDMs with infinite support). Additionally, the proposed approach to estimate the difference between two empirical distributions is not limited to the use of JS measure [179].

11.3 Assessing the Impact of CG-NAT

This section completes the description of the methodology designed to understand the implications of CG-NAT deployment. We here describe the performance metrics used to assess the impact of CG-NAT on users' Web traffic. We leverage indicators collected at different network layers, i.e., network, transport and application layers, and focus on those relevant for Web traffic, e.g., time needed to establish a connection towards remote servers, latency between client and server, time needed to fetch HTTP objects, etc.

11.3.1 Key Performance Indicators

We define the performance indicators we are interested in when considering the CG-NAT impact. Among the many measurements provided by Tstat, we consider for each traffic flow: (i) The TCP RTT between the probe and

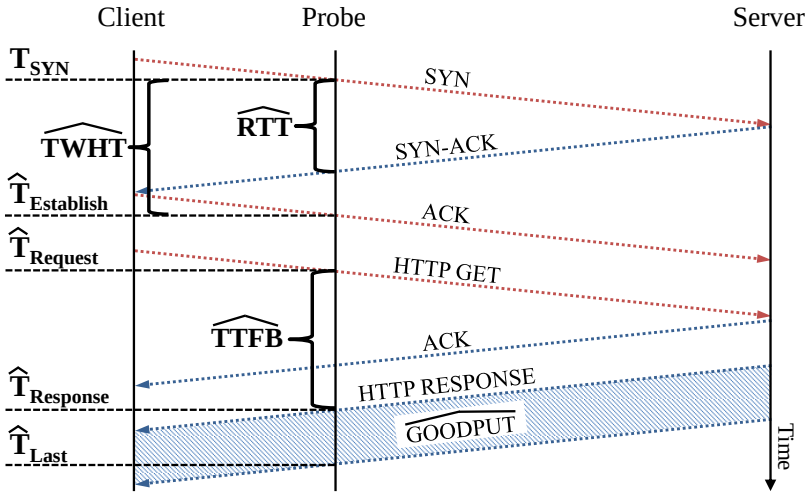


Figure 11.4: An example of HTTP transaction with metrics considered for our analysis.

server; (ii) the Time to Live (TTL) seen at the probe of packets sent by the server; (iii) the total per-flow amount of bytes sent and received by the client; (iv) the application layer protocol (e.g., HTTP, HTTPS); and (v) the timestamps of packets that are instrumental to obtain further indices. Notice that the probe measures the timestamps at a vantage point close to the customers, therefore, for a given metric X we can only gage its estimated measure $\widehat{X}$. Finally, we use server FQDN [32] to split traffic according to the service generating it.

Fig. 11.4 shows Tstat observing a HTTP transaction. In a nutshell, Tstat correlates TCP data segments and acknowledgments, and records timestamps of significant packets. For instance, by correlating times of a data segment with the corresponding acknowledgment, it computes a sample of the RTT. Average and standard deviation of RTT is then obtained by considering all samples in the TCP flow. Since CG-NAT may affect performance at different layers, we detail how we combine basic metrics provided by Tstat to build higher level measurements that we use to contrast the impact of CG-NAT. We classify considered metrics into three main groups, namely network, transport and application layer metrics.

11.3.2 Network Layer Metrics

- **Hop count from Server to PoP** ($\#Hops$).
  The minimum number of hops being traversed by packets transmitted

from the server to the client. The server OS sets the initial value of the TTL, with power of 2 values being the typical choice. Each router along the path then decreases the TTL. The values observed at the probe is thus an indication of the number of hops on the path from the server to the probe located in the PoP. In more details, given a flow, we take the maximum server-to-client TTL observed by Tstat. We then choose x as the exponent minimizing $\#Hops = 2^x - TTL,\ \#Hops > 0$. The resulting $\widehat{\#Hops}$ is the minimum number of hops that packets in the considered flow have traversed before reaching the probe. In our scenario we expect packets received by private home routers to traverse a possibly larger number of hops due to the presence of the CG-NAT (one or more hops).

- **PoP to Server Round Trip Time (RTT).**
  The average RTT Tstat measures in a flow ($\widehat{RTT}$) on packets transmitted from the client to the server. Referring to Fig. 11.4, we consider only the RTT from the probe in the PoP to the server and backward, thus including only the backbone part of the path and ignoring the access portion. We expect packets transmitted by private home routers to experience a higher latency because of the CG-NAT packet processing.

11.3.3 Transport Layer Metrics

- **TCP Three-Way Handshake Time ($TWHT$).**
  The amount of time measured by Tstat ($\widehat{TWHT}$) required to successfully establish a TCP connection using the three-way handshake. Referring to the upper part of Fig. 11.4, let $\hat{T}_{SYN}$ be the timestamp of the SYN packet sent by the client to establish the connection, and let $\hat{T}_{Establish}$ be the timestamp of the packet carrying the ACK message ending the three-way handshake. We define the $\widehat{TWHT}$ as

  $$\widehat{TWHT} = \hat{T}_{Establish} - \hat{T}_{SYN}$$

  In our scenario we expect the $\widehat{TWHT}$ to be higher for private home routers due to the time needed by the CG-NAT to allocate the resources for the new communication session.

For the sake of completeness, we also consider some advanced specific TCP metrics that are directly computed by Tstat [28]:

- **The number of SYN messages during connection setup, $\#SYN$;**

- **The number of out of sequence segments,** $\#OoS$;
- **The number of duplicated segments** $\#Dup$.

These are measurements that we expect to be altered in case of connectivity issues introduced by the CG-NAT. A large value of $\#SYN$, for instance, indicates that the client experienced difficulties in establishing the connection due to, e.g., exhaustion of NAT resources.

11.3.4 Application Layer Metrics

- **Time to first byte (**$TTFB$**).**
  Referring to Fig. 11.4, the amount of time that elapses between the first segment containing the HTTP request sent by the client ($\hat{T}_{Request}$) to the first segment with payload sent by the server ($\hat{T}_{Response}$). We define the $\widehat{TTFB}$ as

  $$\widehat{TTFB} = \hat{T}_{Response} - \hat{T}_{Request}$$

  In HTTP flows, it represents a measure of the time span between the application request issued by the client and the consequent response by the server. Also in this case, we expect the CG-NAT to eventually delay the response time due to NAT operations.

- **Per-connection Goodput (**G**).**
  The average rate at which the server delivers information to the client. This is the paramount performance index for download services. Let $\hat{T}_{Response}$ and $\hat{T}_{Last}$ (see Fig. 11.4) be the timestamps of the first and the last data packet sent by the server, and let D_{down} be the size of the application payload sent by the server. We define the download goodput as

  $$\hat{G}_{down} = \frac{D_{down}}{\hat{T}_{Last} - \hat{T}_{Response}}$$

  It is similarly possible to evaluate the goodput in the upload direction by considering the amount of bytes sent by the client to the server (D_{up}) and referring to the timestamps relative to the client traffic. To avoid the bias of short-lived flows, we evaluate the download goodput only on flows for which $D_{down} \geq 1\,\text{MB}$, and the upload goodput for flows where $D_{up} \geq 500\,\text{kB}$.

Table 11.2: Jensen-Shannon divergence for considered metrics and different Internet services.

Metric	Web Traffic			
	All Flows	www.google.com	TOP-50 Google	phobos.apple.com
$\widehat{\#Hops}$	**0.223**	**0.666**	**0.682**	**0.689**
$\widehat{RTT}$	0.001	0.006	0.007	0.007
$\widehat{TWHT}$	0.002	0.010	0.011	0.016
$\#SYN$	<0.001	<0.001	<0.001	<0.001
$\#OoS$	<0.001	–	–	–
$\#Dup$	0.001	0.001	0.001	<0.001
$\widehat{TTFB}$	0.002	0.006	0.008	0.006

11.4 Impact of CG-NAT on Users' Traffic

The goal of this section is to check whether one of the two classes of customers experiences worse performance than the other due to the type of IP address they have at their home router. To do so, we split flows into two subsets, based on if they are coming from private or public home routers. For each subset, we then compute the empirical PDF for each metric, and we finally evaluate the JS among the two PDFs. We remind the two thresholds identified in Sec. 11.2.2 for the JS: $Q^- = 0.02$ and $Q^+ = 0.1$.

11.4.1 Impact on Network and Transport Layer Metrics

We start our analysis by gaging the impact of CG-NAT on network- and transport- layer metrics described in Sec. 11.3.2 and Sec. 11.3.3, respectively. We report the collected results in Tab. 11.2. We focus on the Web traffic first, as reported on the left-hand side of the table. We show the result of experiments considering flows directed to (i) any remote server ("all flows"); (ii) "www.google.com" servers (i.e., *Google Search*); (iii) TOP-50 most used IP addresses of Google servers ("TOP-50 Google"); and (iv) "phobos.apple.com" servers providing *iTunes Store* content.[3]

As shown, the only metric that consistently overcomes the alarm threshold Q^+ is the number of hops, $\widehat{\#Hops}$, which is highlighted in bold in the

[3]We focus on this selection of services as they appear to be popular on the monitored network, and the amount of TCP flows for each of them satisfies the requirements for a proper use of the JS.

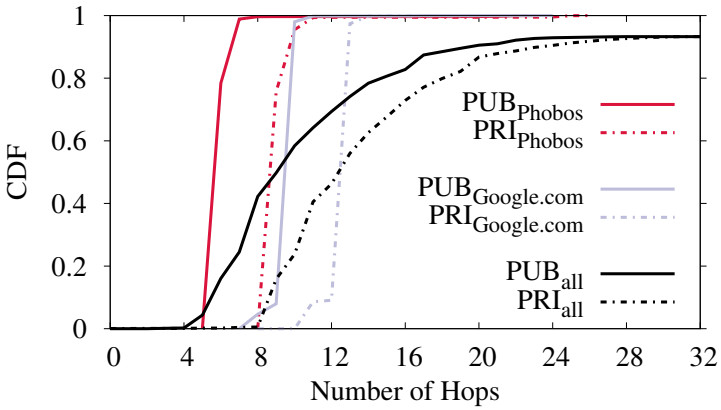


Figure 11.5: CDFs of $\widehat{\#Hops}$ from the server to the client for private and public home routers against different Web services. Clear differences appear.

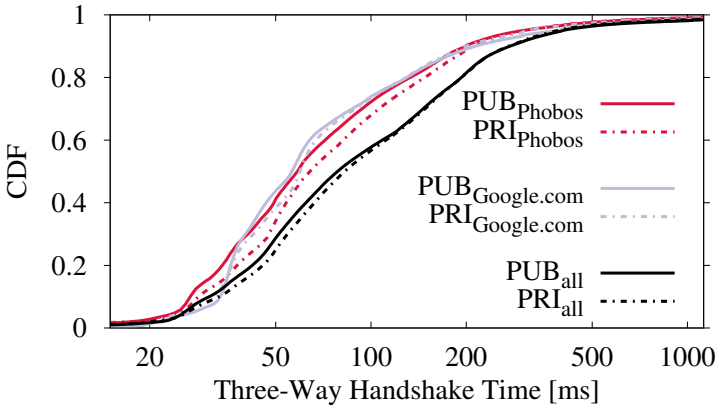


Figure 11.6: CDFs of $\widehat{TWHT}$ for private and public home routers against different Web services. No significant differences are visible.

table. To validate the above finding, we directly compare the distributions of $\widehat{\#Hops}$ in Fig. 11.5. For the ease of visualization, we report the CDF of some services only, as results are similar for others as well. A clear offset between the $\widehat{\#Hops}$ of private and public home routers appears, showing that private ones have to traverse more hops to reach the Internet. Such offset is present for all services. We verified this outcome with the ISP network administrators, who confirmed that the difference is due to some extra routers that packets sent/received by private home routers have to traverse to reach the CG-NAT. However, such routers are well dimensioned and not congested, with no implication on the performance, as testified by other metrics in Tab. 11.2 In summary, the JS values for Web traffic are below Q^{-},

Table 11.3: Jensen-Shannon divergence for goodput distributions in download ($\hat{G}_{down}$) and upload ($\hat{G}_{up}$) directions.

	Service	FQDN	JS_{div}
Download	All	*	0.001
	Facebook Video	fbcdn-video-*.akamaihd.net	0.004
	Tumblr	media.tumblr.com	0.021
	Phobos	phobos.apple.com	0.022
Upload	All	*	0.004
	Amazon S3	eu-irl-*.s3.amazonaws.com	0.007
	Whatsapp	mm*.whatsapp.net	0.033
	Dropbox	dl-*.dropbox.com	0.046

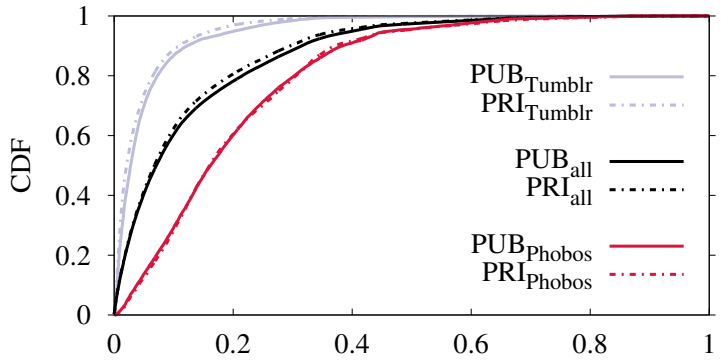
meaning that the CG-NAT configuration of our scenario does not induce any significant bias.

Let us focus on the time needed to establish a TCP connection $\widehat{TWHT}$. This is a typical metric one would expect to be affected by additional delay introduced by the CG-NAT when private home routers try to establish new connections. Indeed, the CG-NAT may require some time to initiate the session and translate addresses. Also in this case JS is very small for Web traffic. Fig. 11.6 shows details distributions for private and public home routers with respect to the same Internet services. Differences are practically negligible. Such result is also confirmed by Tab. 11.2, which reports low values of JS.

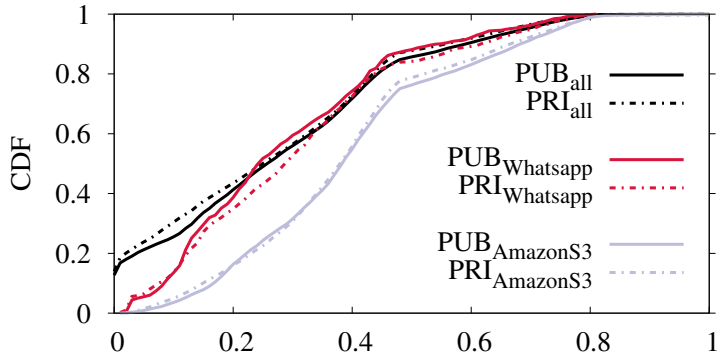
11.4.2 Impact on Application Layer Metrics

We complement the above findings by applying the JS on the indices presented in Sec. 11.3.4. The last row of Tab. 11.2 shows the JS of the Time to First Byte, $\widehat{TTFB}$. Results for Web traffic indicate that this metric is again not affected by the presence of the CG-NAT and that users accessing the Internet from private or public home routers face similar delays.

Next, we perform the same analysis for the Web traffic goodput $\hat{G}$. We consider several popular services that exchange a large amount of data, and for which $\hat{G}_{down}$ is thus relevant, i.e., Facebook Video, Tumblr, and Phobos. For $\hat{G}_{up}$ we selected Amazon S3, Whatsapp, and Dropbox. We report the results in Tab. 11.3, and draw the CDFs in Fig. 11.7.



(a) Download CDFs for flows carrying ≥ 1 MB.



(b) Upload CDFs for flows carrying ≥ 500 kB.

Figure 11.7: Normalized goodput $\hat{G}$ CDFs for flows carrying Web traffic.

Observe that the JS never overcomes the Q^+ threshold, meaning that the CG-NAT does not significantly harm the download/upload speed of private home routers. However, the JS values for Whatsapp and Dropbox in the upload direction, and for Tumblr and Phobos in the download direction, are higher than the Q^- threshold. Fig. 11.7(a) details the distribution of $\hat{G}_{down}$ (we omit Facebook Video to ease the visualization). The curves referring to private and public home routers show indeed very similar trends, justifying small JS values, as confirmed by Tab. 11.3. Fig. 11.7(b) reports results for $\hat{G}_{up}$. Also in this case the curves show very similar CDFs with the only exception of Whatsapp. In this latter case, the difference between the two distributions is confirmed by the JS_{div} =0.033.

Interestingly, a relatively large amount of flows (13.98%) in Fig. 11.7(b) show almost zero throughput. By double-checking, we realize that these are long-lived flows with a duration higher than 10 min, and showing a number of uploaded bytes that slightly exceeds the 500 kB threshold. Indeed, clients

establish a single TCP connection with the remote server and keep sending tiny portions of data intermittently, de-facto zeroing the upload throughput.

11.5 Conclusions

This chapter focused on the impact of Carrier Grade NAT (CG-NAT) on Web browsing experience. CG-NAT are middle boxes deployed by ISPs to mask entire subnetworks by providing customers' routers with a private IP address instead of a public one. This allows ISPs to save on the amount of required public IP addresses and, in turn, on the costs of maintaining them.

We leveraged passive measurements and considered a large dataset of traffic traces that we split according to the type of IP address assigned to users' home routers, i.e., public or private. We then considered multiple performance indicators (e.g., the achieved goodput, the number of lost or out-of-sequence packets, etc.) to compare the two populations. To do so, we relied on the Jensen-Shannon Divergence (JS), a statistical tool to quantify the difference between pairs of distributions, to quickly pinpoint performance metrics showing stochastically significant differences.

We also took good care to associate the output values of the JS to a semantic meaning, i.e., to understand whether the entity of the gap between the two distribution could be the clue of a significant degradation in connectivity performance and, consequently, on QoE for one of the two populations of users. To calibrate the JS sensitivity, we analyzed its behavior on well-known distributions and defined thresholds according to which the difference in performance is negligible, noticeable, or significant.

According to the collected results, CG-NAT can be considered a stable and mature technology that, if properly engineered and configured, does not harm WebQoE. For the considered performance metrics, the difference between distribution was always very small, owing to negligible differences. Only for the number of hops traversed, the JS values fell in the significant range. This is due to a longer path traversed by packets from/to private IP addresses. Such path is however confined inside the ISP network and all the traversed hops are well dimensioned, with negligible impact on performance.

Summary

In this part of the thesis, we addressed the problem of Quality of Experience (QoE) in the Web domain. Recent industry reports show the relevance of WebQoE and how it can impact the value of business. In addition, the introduction and the standardization of new protocols, such as HTTP/2 and QUIC, is bringing new interest in WebQoE assessment.

We started our investigation with a comprehensive report on existing techniques to objectively measure WebQoE. These are in most cases based on the realization of a single event, e.g., time to receive the first byte of payload, time to parse the DOM, etc. Few more refined metrics, e.g., Google's SpeedIndex (SI), focus on the visual progression of the page while it is being loaded, but are extremely demanding in computational resources to the extent of inflating the time needed to load the page. To solve this issue, we introduce two new metrics as generalization of the SI and run experiments on the top-100 Alexa Web pages. We highlight the order relationships existing among metrics and quantify the inflation caused by the SI computation, showing instead how the newly introduced metrics maintain the same relevance at a much lower computational cost.

The second milestone in WebQoE understanding is given by the collection of MOS points provided by actual Internet users. For the purpose, we design a testbed where we can control experiment parameters and we ask volunteers to watch actual Web pages being served over different protocols (i.e., H1, H2) and under different network conditions. Collected results are puzzling: While H2 sensibly reduces the time needed to load a toy page, ultimately improving WebQoE, differences between H1 and H2 with real-life Web pages are far from being clear. Some objective performance indicators show H2 to be the winner (even though with a small margin), while collected MOS grades show users' preference for H1.

Finally, we complement the analysis by considering the impact that middle boxes deployed in networks might have on Web traffic, specifically focusing on the Carrier Grade NAT, given its popularity among ISPs. To do so, we leverage data passively collected in an operational network where users' home

routers are provided either with a private or with a public IP address. We then define performance indicators and identify statistical means to compare the two populations of users. Results show that CG-NAT is now a stable technology that has no impact on Web traffic when properly dimensioned and configured. The performance achieved by users with private address are indistinguishable from those experienced by users with a public IP.

Chapter 12

Conclusions

12.1 Summary and Contributions

In the complex scenario of today's Internet, traffic measurements play a central role to improve network and application design, to guide traffic management, and, in general, to understand Internet dynamics. All players involved, ranging from Internet Service Providers (ISPs) and content providers to regulation agencies, identify traffic monitoring and characterization as the key task for network administration and service delivery enhancement. Punctual network monitoring enables the success of various tasks including the identification of usage patterns and trends, the optimization of the network to support a new service, the detection of malicious behaviors related to malware propagation or cyber attacks, etc.

In addition, the humongous amount of professional activities and personal interactions carried out over the network stress the importance of end users in the Internet paradigm. Increasing attention is indeed attributed to the Quality of Experience (QoE) provided to final users, to their satisfaction in using Internet services, and to secure their sensitive information from malicious adversaries. This thesis contributes in this exact direction by collecting and understanding network traffic exchanged with users.

In this work we achieved an all-round characterization of users' traffic being produced, i.e., *outbound* traffic, consumed, i.e., *inbound* traffic, or unwillingly exchanged with malicious adversaries, i.e., *unsolicited* traffic. We did so by focusing our attention on specific research topics for each of the three traffic categories. Respectively, we targeted: (i) The understanding and the benchmarking of Personal Cloud Storage Services; (ii) the automatic detection of malicious activities and Internet threats; and (iii) the evaluation of QoE in the Web domain.

To achieve our goals, we followed a measurement-based approach that has its foundations in passive data collection from live operational networks, as well as in active probing of Internet services. The former allowed us to continuously monitor the network by simply observing traffic flowing, while with the latter we ran specific and customizable tests. We combined both techniques to achieve complete visibility on network activities.

In the following, we briefly summarize the most relevant findings and contributions for the topics tackled.

In the context of Personal Cloud Storage (Part I), we characterized the typical usage of cloud storage services by means of passive measurements. The collected results show that storage flows are usually small-sized, hinting for the synchronization of files of few bytes or for incremental updates of already-existing contents. In addition, a dependency between the type of device used and the produced workload emerges, suggesting for mobile devices using cloud storage space to backup photos and videos.

We completed the above findings by contributing an automated methodology with which is it possible to benchmark cloud storage providers using customized workloads. We ran extensive batches of experiments using either (i) specific workloads aimed at detecting advanced capabilities eventually implemented in cloud storage clients or (ii) real-life workloads to assess cloud storage performance in realistic conditions. Collected facts highlight a variety of (i) implementation choices, resulting in either clients equipped with various advanced capabilities or simplistic ones, (ii) deployment approaches, with services relying on centralized, partially distributed, or completely distributed architectures, and (iii) provided performance, strictly dependent on the workload used. Also, these three aspects showed to have a strong relationship among them. Sophisticated clients, i.e., clients implementing advanced capabilities, have to be preferred when synchronizing complex workloads. Services with a lightweight client, reacting fast to changes, show good performance with small or medium workloads. Finally, the distance to the data center is not the only parameters to account for. Indeed, a lower latency does not always counterbalance the lack of advanced features or poorly implemented synchronization protocols.

As for Network Security Threats (Part II), we detailed the typical behavior of malware as seen in residential network. For this purpose, we leveraged the knowledge provided by a reference Intrusion Detection System (IDS), which triggered alarms for network activities matching its internal detection rules. Starting from the evidence provided by the IDS, we looked for suspicious and recurrent patterns in the network activity of infected hosts.

Our findings highlighted a complex picture, with threats showing different network behaviors, causing the IDS to fail detecting some of them.

We aimed at augmenting the visibility on malicious attacks and at increasing the detection of Internet threats by contributing MAGMA, an automatic system able to detect network activities that are part of malicious attacks and, in turn, classify the extracted information as malicious or benign. MAGMA is based on a two-steps methodology. In the first instance, MAGMA starts its analysis from network events that triggered an alarm by the IDS. We call these events *malicious seeds*. It then looks for other network activities that are highly correlated with the seed in order to uncover additional elements part of the same attack. The process results in a compact, yet telling, graphical representation of the activities belonging to the same attack. We call the output picture *Network Connectivity Graph (CG)*. The CG facilitates the job of the security analyst, providing her with a complete and understandable picture, rather than with the atomic information of the IDS. The second step of MAGMA consists in a supervised classifier which is able to distinguish between malicious and benign CGs. We leveraged classification techniques based on decision trees and trained the classifier on a set of CGs comprising various network attacks. This guarantees for MAGMA to be a widely-applicable tool and a generic malware classifier, not tailored to a specific family of threats. The collected results showed the effectiveness of MAGMA in identifying additional events related to the malicious activity, as well as its ability to classify CGs with a very high accuracy. More importantly, MAGMA showed to be applicable to a wide spectrum of Internet threats and classes of malware without losing significance or reducing its classification accuracy.

Finally, for User Quality of Experience (Part III), we started our investigation providing a survey on current practices to evaluate users' QoE in the Web domain. We realized that both the industry and the academia refrain from using metrics that consider the whole series of events characterizing the loading process of a Web page, preferring instead to rely on the realization of a single event (e.g., DOM, PLT). However, given the complexity of today's Web pages, we stressed the fact that current practices are limitedly representative for the actual users' WebQoE.

To mitigate this problem, we characterized the most popular metrics over the top-100 Alexa Web pages, highlighting their order relationship, merits, and limitations. In addition, we defined two new metrics inspired by Google's SpeedIndex, which maintain the same perceptual properties at a much lower computational cost. More importantly, we contributed a dataset of over 4,000 Mean Opinion Score (MOS) points, i.e., users' subjective feedbacks. To collect MOS grades, we engineered a testbed allowing us to serve real

Web pages in a controlled environment and asked volunteers to participate in the data collection campaign. Our testbed enabled us to tune network parameters, application protocol used for content delivery, domain sharding, etc., thus verifying Web performance in a variety of scenarios. We employed the testbed to contrast the performance of the newly-introduced HTTP/2 (H2) against HTTP/1.* (H1), as well as to assess te consistency of users' MOS grades with objective metrics (i.e., DOM, PLT). The collected results do not allow the election of a single winner between the two protocols, with objective performance and subjective perception greatly varying according to the Web page and the network parameters configuration.

To complete the picture on WebQoE, we also targeted the impact of Carrier Grade NAT (CG-NAT) devices on Web traffic. CG-NAT is a technology deployed by ISPs as it allows the hiding of entire sub-networks by assigning customers' routers a private IP address instead of a public one, ultimately saving on the costs of maintaining public addresses. We assessed the impact of CG-NAT by splitting users in two groups according to the type of IP address provided and by contrasting the performance achieved by the two populations. To specifically target Web traffic, we defined performance indicators tailored to Web transactions (e.g., time to establish a TCP connection, time to receive the first byte of HTTP payload, etc.) and identified statistical means to quantify the difference between pairs of empirical distributions produced by users with a public/private IP. According to the collected results, CG-NAT does not impact users' experience as its implications on Web traffic performance are negligible. CG-NAT is thus a stable technology, however, ingenuity is required to properly configure and dimension its deployment.

12.2 Future Work

For what concerns the further development of this work, we here propose several directions extending the scope of results achieved so far.

In Part I – Personal Cloud Storage, we presented a methodology to benchmark 11 services and identify the advanced capabilities they implement. The methodology is generic and suitable for being applied to any cloud storage provider. However, the post-processing of packet captures is assigned to service-specific scripts. That is, in order to identify relevant events (e.g., begin/end of upload/download) and compute performance metrics, it is needed to understand how each service tackles the synchronization problem and how users' file are exchanged with the cloud. This requires the execution of toy experiments and some reverse engineering of the service under test. It is

straightforward to understand that a service-specific approach hardly scales when tens of providers are considered or in case services update synchronization protocols frequently, breaking post-processing scripts.

To tackle this issue, an interesting research direction lies in the usage of machine learning techniques for the automatic identification of relevant events in the network traffic. Such events would play the role of *markers* for specific operations carried out during the synchronization procedure, in turn helping in the detection of storage phases (e.g., upload, propagation, etc.). Several works are available in literature targeting Internet traffic identification and classification [23,31] or methods to automate the reverse engineering of protocols [180]. However, they are not specialized for cloud storage traffic and do not provide the degree of granularity required to extract accurate performance figures. The outcomes of additional investigations might fully automate the proposed methodology, enabling interested users to benchmark any cloud storage provider without further interventions by domain experts.

In Part II – Network Security Threats we engineered a system, MAGMA, able to automatically identify network events related to a specific activity (i.e., a seed) and to classify the extracted knowledge as malicious or benign. MAGMA looks for correlations over time and over multiple hosts to identify network events which are part of the seed activity. To do so, it requires at least three observations of the seed to properly characterize it.

In order to collect the three observations in a reasonable amount of time, MAGMA should operate on a large network where multiple hosts exchange traffic. This calls for additional investigations on the scalability of the proposed solution. The main goal of our research was to study the applicability of the ideas driving MAGMA design and to assess the achievable results, instead of provisioning an implementation for production-ready scenarios. In this latter case, indeed, additional effort is required to eventually re-implement some components of the system and make it powerful enough to sustain traffic analysis with real-time constraints.

Additionally, MAGMA classification output is, by now, limited to two classes, i.e., malicious or benign activities, and does not offer a finer granularity on the network events being analyzed. It would be interesting to work in this direction, aiming at the design of a classifier comprising multiple output classes. MAGMA should then be able to provide additional details on the analyzed activity and, more specifically, particularize malicious threat types, e.g., botnets, DDoS attacks, trojan horses, etc.

In Part III – User Quality of Experience, we focused on how to evaluate the QoE for Web users and on the impact that recent protocols, i.e., H2, and

middle boxes, i.e., CG-NAT, have on Web traffic. While results for CG-NAT undoubtedly affirm the negligible influence of such technologies on users' QoE, the picture emerging from the performance assessment of HTTP/2 is not as much clear, requiring additional efforts for a thorough understanding.

In this context, further analysis is required on the practices to measure WebQoE. In Sec. 10.1 we reported a thorough survey on QoE indicators, highlighting merits and limitations of current practices. More interestingly, we introduced two new metrics, namely ByteIndex and ObjectIndex, which generalize Google's SpeedIndex, embedding a high significance of WebQoE at a negligible computational cost. These two metrics can be further optimized to estimate QoE more closely, opening for interesting future directions:

- **Closer SpeedIndex approximation**
  {Byte, Object}Index metrics provide optimistic lower bounds to the SpeedIndex. This is due to the fact that their completion increases upon reception of any bytes/objects, including (i) those that are not painted (e.g., scripts) and (ii) those that take time to render (e.g., alpha images, complex CSS). Conditioning over content type (e.g., null weight for scripts) would cope with (i), while considering execution times (e.g., no completion increase before DOM, estimation of time from reception to paint) could address (ii).

- **Psycho-behavioral model: Content bias**
  Extending the above reasoning, it could be argued that taking explicitly into account object type or size could be worth investigating. For instance, for some object types a logarithmic reward can be expected from their byte-wise size (e.g., size of a JPEG image encoded with higher quality may significantly increase, but the added value is likely sub-linear). Similarly, it may be argued that users perceive advertisement with a different value than content, which could be factored in by defining a weight function $w_i = 1 - \mathbb{1}_{\text{AdBlock}(i)}$ with $\mathbb{1}_{\text{AdBlock}(i)} = 1$ whenever the domain name of object i belongs to the AdBlock list [181].

- **Large-scale study**
  An obvious improvement of this work could be to extend the characterization we conducted over the Alexa top-100 dataset by either (i) considering pages beyond the top-100, or (ii) performing the same experiment employing geographically-dispersed vantage points (e.g., M-Lab/PlanetLab nodes, Amazon Elastic Compute Cloud instances, etc.).

Also, perceived differences between H2 vs H1 and their relationship with objective metrics (e.g., DOM, PLT, etc.) constitute a topic for further investigations. Indeed, according to the collected results, H2 unleashes its power

only with a toy page where the new protocol features can be exploited to reduce the Page Load Time (PLT), consequently improving WebQoE. This outcome, however, does not hold with real-life pages, where H2 performance are not consistent from both an objective and a subjective perspective. Indeed, while objective metrics show a little advantage of H2, Web users give their preference to H1. Thus, it would be of undoubted interest to collect additional MOS datasets in order to gather clearer results, as well as to better account for the relationship between objective metrics and users' feedback.

Finally, another important aspect to explore is the dependency between protocol performance, page characteristics, and connection properties. The amount of retrieved objects, the number of contacted domains, the existence of JavaScript triggering the download of additional content, the available bandwidth, etc. are all factors impacting the page construction process. An interesting goal would be the selection of the most suitable protocol, i.e., the best performing one, according to scenario faced. This calls for the use of machine learning techniques and for the modeling of decision trees. A fundamental aspect of this step lies in the selection of features representative for the considered scenario. These should involve characteristics belonging to: (i) The Web page itself (e.g., number of objects, number of domains, object sizes, object types, etc.); (ii) the path from the user to remote servers (e.g., latency, loss, bandwidth); (iii) the user context (e.g., device being used, screen size, connection properties, etc.)

Part IV

Appendixes

Appendix A

MAGMA

List of Classification Features

Tab. A.1 list all features extracted from Seed-CGs divided by categories. Most of them are self-explanatory. The *CDN hostname ratio* takes into account the presence of CDN in Web pages. We assume that if a hostname is linked to more than 3 distinct IP addresses, the content it refers to is hosted on a CDN. We define such hostname a *CDN hostname*.

Table A.1: Full list of features extracted from Seed-CGs. Underlined features are selected by mRMR.

Topology	
num. of	total number of nodes
num. of	total number of edges
num. of	failed DNS queries events
num. of	object-path nodes
num. of	hostname nodes
num. of	serverIP nodes
num. of	UDP-ports nodes
num. of	TCP-ports nodes
num. of	DNS error types
num. of	nodes with single edge
min/max/avg/std	in-degree object-path nodes
min/max/avg/std	in-degree hostname nodes
min/max/avg/std	in-degree serverIP nodes
min/max/**avg**/std	out-degree object-path nodes
min/max/avg/std	out-degree hostname nodes
min/max/avg/std	out-degree serverIP nodes
ratio	giant connection ratio
ratio	ratio num. of CDN hostnames[†] over total hostnames (*CDN hostname ratio*)

HTTP	
num. of	requests per each method [GET, POST, others]
num. of	replies per each response status [20x, 30x, **40x**, 50x]
num. of	replies per each content-type [text, image, application, binary, multipart, multimedia]
num. of	distinct user-agent strings
min/max/avg/std	user-agent strings length
min/max/**avg**/std	requests per distinct user-agent string
min/max/avg/std	user-agent strings blank chars

Syntax	
num. of	hostname nodes being IP addresses
num. of	hostname starting with `www`
min/max/avg/std	hostname string length
min/max/**avg**/std	hostname digits and alphabetic chars ratio
min/max/avg/std	for hostname up to 2nd LD, num. of distinct 3rd LD

Occurrences	
min/**max**/avg/std	object-path nodes events
min/max/avg/std	hostname nodes events
min/**max**/avg/std	DNS fail nodes events
min/**max**/avg/std	DNS succeed nodes events
min/max/avg/std	TCP port nodes events
min/max/avg/std	UDP port nodes events

Appendix B

WebQoE

B.1 Page Catalog

Here we present a thorough analysis of the Web pages catalog. Web pages macro features (e.g., page size, number of objects, number of contacted domains and RTT towards such domains) can be contrasted at a glance in Fig. 10.6. The following tables are instead intended for a more detailed view of Web pages, focusing on objects being retrieved, the related amount of bytes, and network features like the number of domains contacted and the amount of connections opened to fetch all the required objects. Specifically:

- Tab. B.1 reports statistics on the objects composing each Web page. It provides the total number of objects fetched to build the page together with the breakdown per object types (e.g., HTML, Images, etc.). The ratio between each type and the total amount of objects is reported within brackets;
- Tab. B.2 shows the overall amount of downloaded bytes, together with the mean and standard deviation of bytes in a per-object fashion. A breakdown of retrieved bytes per object type (as defined in Tab. B.1) is provided, as well as the ratio with respect to the total amount of retrieved bytes;
- Tab. B.3 highlights the number of contacted domains and established connections, reporting the mean and the standard deviation of objects and bytes retrieved from each domain / over each connection.

Table B.1: Web pages characterization – Objects details. For each Web page we report (i) the total number of objects; (ii) the number of objects per type (e.g., HTML, images, css, javascript); and (iii) the ratio of each type over the total number of objects.

Web page	**Total Objects**	**Object Types** *(Ratio)*									
		HTML		Images		Style		Scripting		Other	
Synthetic Web page	361	1	*(0.00)*	360	*(1.00)*	0	*(0.00)*	0	*(0.00)*	0	*(0.00)*
pole-emploi.fr	71	2	*(0.03)*	37	*(0.52)*	5	*(0.07)*	0	*(0.00)*	27	*(0.38)*
booking.com	67	1	*(0.01)*	46	*(0.69)*	6	*(0.09)*	10	*(0.15)*	4	*(0.06)*
free.fr	32	2	*(0.06)*	10	*(0.31)*	8	*(0.25)*	7	*(0.22)*	5	*(0.16)*
lcl.fr	103	3	*(0.03)*	66	*(0.64)*	10	*(0.10)*	23	*(0.22)*	1	*(0.01)*
societegenerale.fr	21	1	*(0.05)*	13	*(0.62)*	1	*(0.05)*	4	*(0.19)*	2	*(0.10)*
leboncoin.fr	52	2	*(0.04)*	38	*(0.73)*	1	*(0.02)*	10	*(0.19)*	1	*(0.02)*
amazon.com	213	9	*(0.04)*	180	*(0.85)*	7	*(0.03)*	8	*(0.04)*	9	*(0.04)*
bouyguestelecom.fr	34	1	*(0.03)*	21	*(0.62)*	3	*(0.09)*	6	*(0.18)*	3	*(0.09)*
stackoverflow.com	26	1	*(0.04)*	13	*(0.50)*	2	*(0.08)*	7	*(0.27)*	3	*(0.12)*
dailymotion.com	51	3	*(0.06)*	39	*(0.76)*	2	*(0.04)*	4	*(0.08)*	3	*(0.06)*
laredoute.fr	51	2	*(0.04)*	28	*(0.55)*	1	*(0.02)*	3	*(0.06)*	17	*(0.33)*
leroymerlin.fr	66	2	*(0.03)*	51	*(0.77)*	2	*(0.03)*	9	*(0.14)*	2	*(0.03)*
labanquepostale.fr	65	2	*(0.03)*	30	*(0.46)*	8	*(0.12)*	13	*(0.20)*	12	*(0.18)*
chinadaily.com.cn	212	50	*(0.24)*	133	*(0.63)*	5	*(0.02)*	20	*(0.09)*	4	*(0.02)*
diply.com	106	11	*(0.10)*	75	*(0.71)*	3	*(0.03)*	4	*(0.04)*	13	*(0.12)*
tf1.fr	77	1	*(0.01)*	32	*(0.42)*	2	*(0.03)*	33	*(0.43)*	9	*(0.12)*
imdb.com	130	8	*(0.06)*	80	*(0.62)*	12	*(0.09)*	25	*(0.19)*	5	*(0.04)*
over-blog.com	170	13	*(0.08)*	104	*(0.61)*	9	*(0.05)*	21	*(0.12)*	23	*(0.14)*
leboncoin-iledefrance-.fr	72	4	*(0.06)*	48	*(0.67)*	1	*(0.01)*	14	*(0.19)*	5	*(0.07)*
darty.com	143	7	*(0.05)*	86	*(0.60)*	3	*(0.02)*	29	*(0.20)*	18	*(0.13)*
marmiton.org	180	14	*(0.08)*	106	*(0.59)*	11	*(0.06)*	39	*(0.22)*	10	*(0.06)*
allocine.fr	178	7	*(0.04)*	130	*(0.73)*	3	*(0.02)*	20	*(0.11)*	18	*(0.10)*
bfmtv.com	127	11	*(0.09)*	64	*(0.50)*	3	*(0.02)*	20	*(0.16)*	29	*(0.23)*
commentcamarche.net	200	15	*(0.08)*	117	*(0.59)*	9	*(0.05)*	39	*(0.20)*	20	*(0.10)*

Table B.2: Web pages characterization – Bytes details. For each Web page we report (i) the total number of bytes; (ii) their per-object average and standard deviation; (iii) the number of bytes per object type; and (iv) the ratio of bytes per object type over the total number of bytes.

Web page	**Total KBytes**	**KBytes Mean**	**KBytes Stdev**	**KBytes Per Object Type** *(Ratio)*									
				Html		Images		Style		Scripting		Other	
Synthetic Web page	952	2.64	7.39	3	*(0.00)*	949	*(1.00)*	0	*(0.00)*	0	*(0.00)*	0	*(0.00)*
pole-emploi.fr	1205	16.97	26.57	110	*(0.09)*	498	*(0.41)*	224	*(0.19)*	0	*(0.00)*	373	*(0.31)*
booking.com	1157	17.27	26.41	98	*(0.08)*	378	*(0.33)*	178	*(0.15)*	349	*(0.30)*	154	*(0.13)*
free.fr	724	22.63	38.05	11	*(0.01)*	415	*(0.57)*	20	*(0.03)*	77	*(0.11)*	201	*(0.28)*
lcl.fr	1669	16.20	31.09	84	*(0.05)*	918	*(0.55)*	80	*(0.05)*	572	*(0.34)*	16	*(0.01)*
societegenerale.fr	318	15.13	18.56	6	*(0.02)*	222	*(0.70)*	13	*(0.04)*	71	*(0.22)*	5	*(0.02)*
leboncoin.fr	325	6.24	15.16	38	*(0.12)*	187	*(0.58)*	3	*(0.01)*	95	*(0.29)*	2	*(0.00)*
amazon.com	5714	26.83	42.11	122	*(0.02)*	5294	*(0.93)*	52	*(0.01)*	177	*(0.03)*	68	*(0.01)*
bouyguestelecom.fr	689	20.26	35.04	13	*(0.02)*	318	*(0.46)*	45	*(0.07)*	75	*(0.11)*	237	*(0.34)*
stackoverflow.com	391	15.05	19.22	41	*(0.10)*	188	*(0.48)*	52	*(0.13)*	64	*(0.16)*	47	*(0.12)*
dailymotion.com	815	15.98	19.30	34	*(0.04)*	537	*(0.66)*	70	*(0.09)*	119	*(0.15)*	56	*(0.07)*
laredoute.fr	1055	20.68	26.78	26	*(0.02)*	638	*(0.60)*	44	*(0.04)*	46	*(0.04)*	301	*(0.29)*
leroymerlin.fr	915	13.87	20.68	28	*(0.03)*	618	*(0.68)*	33	*(0.04)*	191	*(0.21)*	46	*(0.05)*
labanquepostale.fr	1086	16.71	27.36	17	*(0.02)*	771	*(0.71)*	62	*(0.06)*	138	*(0.13)*	98	*(0.09)*
chinadaily.com.cn	3387	15.97	37.45	90	*(0.03)*	3133	*(0.93)*	27	*(0.01)*	127	*(0.04)*	10	*(0.00)*
diply.com	6803	64.18	144.69	51	*(0.01)*	6252	*(0.92)*	82	*(0.01)*	85	*(0.01)*	332	*(0.05)*
tf1.fr	1532	19.90	35.12	21	*(0.01)*	1001	*(0.65)*	78	*(0.05)*	298	*(0.19)*	135	*(0.09)*
imdb.com	1590	12.23	14.27	93	*(0.06)*	812	*(0.51)*	136	*(0.09)*	540	*(0.34)*	9	*(0.01)*
over-blog.com	3632	21.36	28.65	93	*(0.03)*	2383	*(0.66)*	128	*(0.04)*	427	*(0.12)*	600	*(0.17)*
leboncoin-iledefrance-.fr	468	6.50	9.33	43	*(0.09)*	209	*(0.45)*	9	*(0.02)*	139	*(0.30)*	68	*(0.15)*
darty.com	2354	16.46	30.12	62	*(0.03)*	1454	*(0.62)*	35	*(0.01)*	228	*(0.10)*	575	*(0.24)*
marmiton.org	2515	13.97	31.38	123	*(0.05)*	1825	*(0.73)*	114	*(0.05)*	315	*(0.13)*	139	*(0.06)*
allocine.fr	2812	15.80	30.16	77	*(0.03)*	1858	*(0.66)*	83	*(0.03)*	239	*(0.09)*	554	*(0.20)*
bfmtv.com	2641	20.79	102.77	43	*(0.02)*	1692	*(0.64)*	74	*(0.03)*	291	*(0.11)*	541	*(0.20)*
commentcamarche.net	3079	15.39	33.43	140	*(0.05)*	1937	*(0.63)*	98	*(0.03)*	610	*(0.20)*	294	*(0.10)*

Table B.3: Web pages characterization – Domains & Connections. For each Web page we report (i) the number of domains contacted; (ii) the number of objects and bytes (average and standard deviation) hosted on each domain; (iii) the number of established connections; and (iv) the number of objects and bytes (average and standard deviation) retrieved over each connection.

Web page	**Domains No.**	**Per-Domain**				**Connect No.**	**Per-Connection**			
		Objects		KBytes			Objects		KBytes	
		Mean	Std	Mean	Std		Mean	Std	Mean	Std
Synthetic Web page	1	361.00	0.00	952	0	6	60.17	1.95	159	5
pole-emploi.fr	3	23.67	27.26	402	355	8	8.88	6.47	151	94
booking.com	4	16.75	13.88	289	231	16	4.19	2.60	72	53
free.fr	5	6.40	9.81	145	219	10	3.20	2.23	72	81
lcl.fr	5	20.60	39.20	334	658	10	10.30	7.71	167	159
societegenerale.fr	5	4.20	5.42	64	84	21	1.00	0.00	15	19
leboncoin.fr	7	7.43	11.10	46	40	15	3.47	2.68	22	26
amazon.com	8	26.63	62.15	714	1810	20	10.65	13.29	286	404
bouyguestelecom.fr	9	3.78	5.94	77	180	15	2.27	1.53	46	62
stackoverflow.com	10	2.60	2.29	39	74	12	2.17	1.72	33	45
dailymotion.com	10	5.10	6.91	81	106	23	2.22	1.89	35	35
laredoute.fr	11	4.64	5.96	96	163	19	2.68	1.92	56	50
leroymerlin.fr	11	6.00	9.87	83	150	25	2.64	2.68	37	54
labanquepostale.fr	13	5.00	11.29	84	265	19	3.42	3.31	57	101
chinadaily.com.cn	16	13.25	27.60	212	572	71	2.99	5.98	48	128
diply.com	17	6.24	15.21	400	1451	26	4.08	4.05	262	459
tf1.fr	17	4.53	7.59	90	172	37	2.08	1.60	41	54
imdb.com	21	6.19	11.85	76	162	35	3.71	3.80	45	69
over-blog.com	23	7.39	10.56	158	372	39	4.36	4.60	93	125
leboncoin-iledefrance-.fr	24	3.00	2.71	20	28	39	1.85	1.87	12	16
darty.com	33	4.33	9.63	71	325	56	2.55	2.93	42	108
marmiton.org	34	5.29	12.66	74	278	66	2.73	3.99	38	114
allocine.fr	41	4.34	5.09	69	139	62	2.87	3.78	45	77
bfmtv.com	45	2.82	3.73	59	205	103	1.23	0.94	26	114
commentcamarche.net	47	4.26	7.36	66	173	86	2.33	3.01	36	74

Appendix C

Impact of CG-NAT

C.1 Additional Results

In Sec. 11.4 we focused our efforts on the impact of CG-NAT on Web traffic. CG-NAT devices, however, affect also other types of traffic, e.g., P2P and unsolicited requests, while the configuration of the NAT itself impacts the amount of resources that can be saved. In this section, we (i) analyze P2P traffic to quantify to which extent peers behind NAT devices are impaired in performance; (ii) assess the amount of unsolicited requests home routers are subject to when provided with a private or public IP address; (iii) quantify the amount of customers inside the monitored ISP PoP running servers that thus require a public IP address; and finally (iv) estimate the potential resource savings according to different address translation policies and configurations.

C.1.1 Analysis on P2P Traffic

We consider traffic generated by BitTorrent, which is the most used P2P application in our dataset, and the households (i.e., peers) showing at least one flow carrying 1 MB of P2P traffic.

Reachability conditions have a relevant weight for P2P applications. To better describe the scenario in which we operate, we recall Fig. 11.2 and expand it to include P2P peers. The result is shown in Fig. C.1.

Three regions can be identified: (i) the monitored PoP; (ii) the ISP network; and (iii) the public Internet. P2P peers can be located either inside the ISP network (α, β in Fig. C.1) or in the public Internet (γ). The ISP assures end-to-end connectivity among customers within its network, independently on the type of IP address assigned to each home router. Instead, reachability is not guaranteed in case one of the two peers is outside the ISP network and its counterpart is equipped with a private IP address.

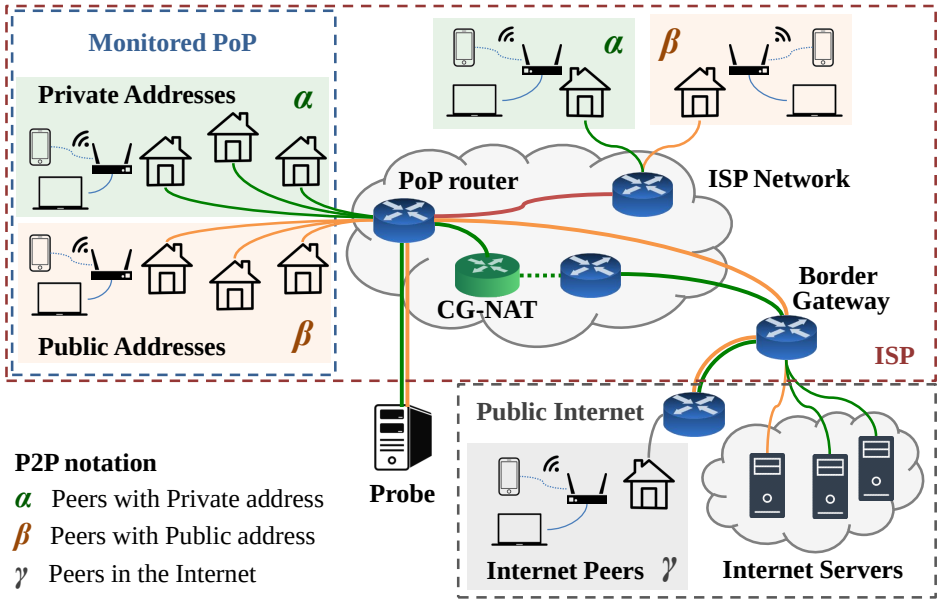


Figure C.1: Monitoring scenario for P2P traffic.

To assess this, we check if peers are able to receive incoming connections. We define a peer as *reachable* if its home router is properly configured and ports are forwarded to the P2P application. In case the home router is misconfigured and the P2P application is non-reachable, we define a peer as *unreachable*. Four classes of peers emerge:

- **Public–Unreachable**: Any peer behind public home routers that does not receive incoming connections;
- **Public–Reachable**: Any peer behind public home routers that receives incoming connections from both the ISP and the Internet. This is the only class of peers that is reachable by everyone;
- **Private–Unreachable**: Any peer behind private home routers that does not receive incoming connections;
- **Private–Reachable**: Any peer behind private home routers that receives incoming connection from other peers in the ISP network (α and β in Fig. C.1). Reachability from peers in the Internet is not guaranteed as the CG-NAT limits incoming connections.

Tab. C.1 characterizes the number of active peers over TCP and UDP according to their reachability condition. Notice that only the 35% (50%) of

Table C.1: Statistics on peers according to their reachability conditions.

		Private	Public
TCP	Reachable	631 (35%)	496 (50%)
	Unreachable	1188 (65%)	499 (50%)
	Total	1819	995
UDP	Reachable	891 (77%)	591 (95%)
	Unreachable	262 (23%)	33 (5%)
	Total	1153	624

peers with a private (public) IP have their home router properly configured and are thus reachable over TCP. This potentially owes to the scarce success of NAT traversal techniques for TCP flows.

In the case of UDP, instead, the ratio of reachable peers is higher both for private (77%) and public (95%) home routers. This is due to the fact that NAT traversal techniques like STUN [182, 183] are more effective over UDP, and enable peers to receive incoming connections.

Key Performance Indicators

All the performance metrics described in Sec. 11.3.1 still hold for the P2P traffic analysis. The only exception consists in the way download or upload speed is evaluated. Indeed, $\hat{G}$ as defined in Sec. 11.3.4 is a representative measure of performance when the download of a content is done using a single flow, e.g., when downloading some software from the web. For P2P applications, however, the speed at which a peer downloads a content is more complicated to compute since multiple parallel connections are used by the application. For instance, BitTorrent typically downloads content from 5 to 10 peers at the same time, using both TCP or UDP at the transport layer. To measure the overall performance of a peer, we compute the average download (upload) throughput $\widehat{Thru}$ considering all data received (sent) by a client in a time interval of duration $\Delta T = 10\,\text{mins}$. Formally, given time interval i, we consider all TCP and UDP flows that Tstat classifies as BitTorrent, and terminated in the time interval, $F(i) = \{f | \hat{T}_{Last}(f) \in i\Delta T\}$. Let $D_{tot}(i) = \sum_{k \in F(i)} D(k)$ the total amount of data those flows carried. We then define the average throughput ($Thru$) as

$$\widehat{Thru}(i) = \frac{D_{tot}(i)}{\Delta T}$$

Table C.2: Jensen-Shannon divergence for considered metrics and P2P traffic.

Metric	P2P	
	Reachable	Unreachable
$\widehat{\#Hops}$	**0.184**	**0.162**
$\widehat{RTT}$	0.055	0.002
$\widehat{TWHT}$	0.029	0.008
$\#SYN$	<0.001	<0.001
$\#OoS$	–	<0.001
$\#Dup$	<0.001	<0.001
$\widehat{TTFB}$	0.031	0.005
$\widehat{Thru_{Down}}$	0.005	0.004
$\widehat{Thru_{Up}}$	0.004	0.003

Impact of CG-NAT on P2P Traffic

We here focus on the JS for BitTorrent traffic, distinguishing between (i) reachable peers (i.e., those who have port forwarding properly configured at their home router); and (ii) unreachable peers. Tab. C.2 shows the results.

Several works in the literature ([154,155]) show how multiple factors impact P2P performance: Content popularity, content availability, type of peers (e.g., seeders, leechers) involved in the transfer, peers cooperation techniques (e.g., tit-for-tat), etc. Let us assume a population of peers in our dataset is not biased by any of these factors. To verify such assumption, we perform 10-fold validation. We randomly split the P2P dataset into 10 sub-groups and compare the JS PDF of each group against all other groups. In none of these experiments we detected statistically significant differences, i.e., JS $> Q^{-}$, validating that our population of peers shows homogeneous performance. This allows us to safely contrast *reachable* and *unreachable* peers.

Let us focus first on the lower part of the table, showing the result for upload ($\widehat{Thru_{Up}}$) and the download ($\widehat{Thru_{Down}}$) throughput. For both classes of peers, the computed values are one order of magnitude below the Q^{-} threshold, proving that the CG-NAT does not affect throughput for BitTorrent. In the nutshell, peers obtain the same performance, no matter if they have private or public addresses.

Focusing on the upper part, instead, we notice that $\widehat{\#Hops}$ produces values higher than the alarm threshold Q^{+}. This is due to the additional

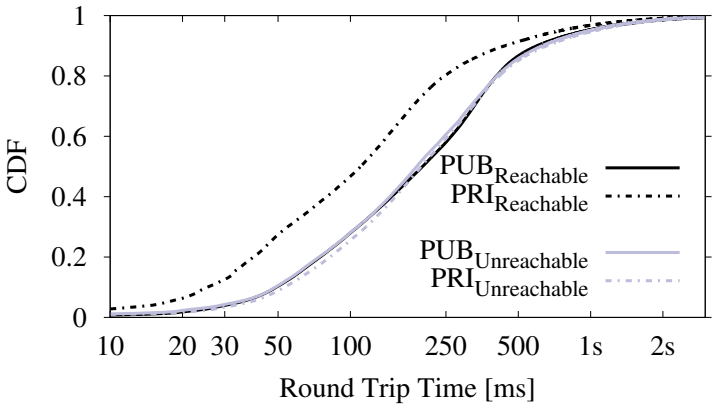


Figure C.2: RTT for P2P traffic according to peers' reachability.

Table C.3: Distribution of contacted peers.

	Reachable	**Unreachable**
Private	58% ISP – 42% Internet	7% ISP – 93% Internet
Public	10% ISP – 90% Internet	6% ISP – 94% Internet

hops that traffic generated by or directed to private home routers have to traverse, as similarly seen in the case of Web traffic (see Sec. 11.4).

More interestingly, JS values are below Q^- for *unreachable* peers, but they fall in the noticeable range ($[Q^-, Q^+]$) for *reachable* peers. Let us consider first the results for $\widehat{RTT}$. Fig. C.2 shows the $\widehat{RTT}$ of BitTorrent connections.[1] It is evident that the RTT for reachable peers with a private IP address appears to be lower than the RTT measured for all the other peers. To better understand this aspect, we characterize the reachability condition of peers inside the ISP and of their counterparts.

To exemplify reachability conditions, refer to Fig. C.1. It depicts peers inside the ISP (α, β) and peers outside (γ). Consider a *reachable* peer with private IP (α). It can receive incoming connections from all the peers inside the ISP network (β), but not from peers in the Internet (γ). On the other hand, a *reachable* peer with public IP (β) can receive incoming connections from both the peers inside the ISP (α) and from peers in the Internet (γ).

[1]The measured $\widehat{RTT}$ is inflated by the queuing delay of packets stacked in the upload queue of home routers. All measurements are equally biased by this phenomenon and it does not harm the reliability of the metric.

Such reachability conditions have implications also on the distribution of contacted peers, as shown by the Tab. C.3, which reports the percentage of contacted peers by *reachable* and *unreachable* peers. Reachable peers with a private IP establish more connections with other peers inside the ISP (52%) than in the Internet (42%). All other classes of peers are more prone to connect to peers in the Internet (>90%). This is due to peers in the ISP that contacted private but reachable peers, like α.

As a consequence, private reachable peers experience a lower RTT since they contact peers inside the ISP network, which are closer in space and exhibit a lower RTT. This also reflects in the other metrics, $\widehat{TWHT}$ and $\widehat{TTFB}$, apparently showing noticeable differences. Being these metrics strictly related to the RTT (see Fig. 11.4), this behavior is expected and is a direct consequence of lower RTT experienced by private reachable peers. Despite this bias, no evident impact is observed in throughput, as shown in Tab. C.2.

C.1.2 Unsolicited Traffic

We here quantify how many home routers interfacing the Internet by means of public/private IP address are exposed to unsolicited incoming traffic. We perform an analysis based on the destination port used, which assesses the number of connection attempts we observe in our failed-TCP logs. First, we compile a list of IP addresses corresponding to potential attackers by counting the number of SYN messages they generate. In particular, we label as attacker every IP address that forges SYN messages directed to 50 (or more) distinct home routers in our PoP. Second, we check the port list and focus on those associated to known services or threats. Hence, for each destination port, we compute: (i) The number of distinct attackers; (ii) the number of home routers contacted; and (iii) the number of connection attempts.

Tab. C.4 reports, for the top-20 most contacted ports, the percentages of private and public home routers inside the PoP being targets of connection attempts. As clearly shown, the number of potential victims in the public home router set is close to 80% for the vast majority of the considered ports. Conversely, these percentages are minimal for private home routers (below 5% in the worst case), as private addresses can be reached only if the counterpart is within the borders of the ISP network. We observe similar results for the amount of connection attempts and the number of distinct attackers. Considering Port 22, for instance, the number of connection attempts peaks at 2 Million against public home routers and stops at 6,500 against private home routers. Similarly, 10,000 attackers are found in the global Internet, while less than 200 are detected inside the ISP. We thus can validate our hypothesis: Public home routers are more exposed to attacks than private

Table C.4: Percentage of public and private home routers targeted by unsolicited traffic. Top-20 destination ports are shown and sorted according to public addresses population.

Port	Description	Private	Public
0	Illegal –OS fingerprinting–	0.1	79.3
135	*Multiple*†, MS Remote Procedure Call⋆	<0.1	79.3
143	ADM†⋆ IMAP	<0.1	79.3
1433	*Multiple*†⋆, MS SQL Server	0.1	79.3
2222	*Multiple*†, Rockwell CSP2	1.0	79.3
3306	Nemog†, W32.Spybot†, MySQL Server	0.1	79.3
3389	Windows Remote Desktop Protocol⋆	0.1	79.3
5900	Evivinc†, Virtual Network Computing	<0.1	79.3
32764	Cisco Access Point⋆, Cisco Routers⋆	<0.1	79.3
3128	*Multiple*†, Proxy servers	<0.1	79.3
22	*Multiple*†, SSH	3.2	79.2
445	*Multiple*†⋆, MS Active Directory	<0.1	79.2
995	POP3 over SSL	<0.1	79.2
8080	*Multiple*†, HTTP Alternate	<0.1	79.1
25	*Multiple*†, SMTP	<0.1	79.0
443	*Multiple*†, HTTPS / SSL	0.1	78.9
80	*Multiple*†, HTTP	1.8	78.5
21	*Multiple*†, FTP	4.6	78.1
23	*Multiple*†, Telnet	<0.1	77.9
139	*Multiple*†⋆, NetBIOS	<0.1	56.6

†: Worm or Threat, ⋆: Known vulnerability.

ones, and CG-NAT represents the first line of defense to limit unsolicited traffic. For instance, the CG-NAT has the potential of curbing the spread of those bots whose goal is to exploit vulnerabilities at the home routers.

C.1.3 Active Servers in the PoP

The per-year cost to rent a public IP address (around 10$) is a non-negligible expense when multiplied for the number of subscribers. Thus, considering the results presented in Sec. 11.4 and in Sec. C.1, we can conclude that the ISP has no actual incentive to provide users' home routers with public IP addresses. However, one may argue that some customers may be interested in having a public IP address to host servers they want to maintain accessible

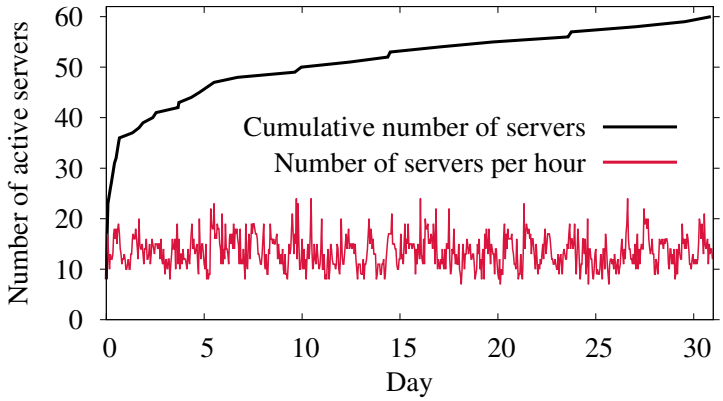


Figure C.3: Number of active servers with per-hour granularity (red curve), and cumulative number of active servers over one month (black curve).

from outside the ISP network. In this scenario, the only way to guarantee the server reachability is to assign a public IP address to the customer's home router. We thus perform a further investigation to gage the number of home routers with public address behind which some kinds of servers are running. To count the number of active servers in our dataset we consider all public home routers that generate at least one HTTP, HTTPS, Internet Message Access Protocol (IMAP), POP3 or Simple Mail Transfer Protocol (SMTP) connection on a daily basis.

Fig. C.3 shows the cumulative number of distinct active servers we observe over one month's time, together with the per-hour number of active servers. This result is boggling. Among all the public home routers we monitor in the PoP, only 60 of them are actually running services being accessed from the Internet. This enforces our claim that the users have no effective need to ask for home routers with public IP addresses.

C.1.4 Potential Saving for Different NAT Policies

In this last section, we aim at providing some practical guidelines for the configuration of CG-NATs. In particular, we analyze different NATing policies and their saving in terms of public IP addresses to be used to offer connectivity to the ISP customers. We consider two different cases: (i) A simple NAT policy according to which a customer is given a public IP address for the period of time she is active; (ii) a NAT and Port Address Translation (PAT) policy for which a customer is given a block of ports on given public IP address, for the time she is active. A *period of activity* of customer starts when the first packet is observed and ends when no packets have been ob-

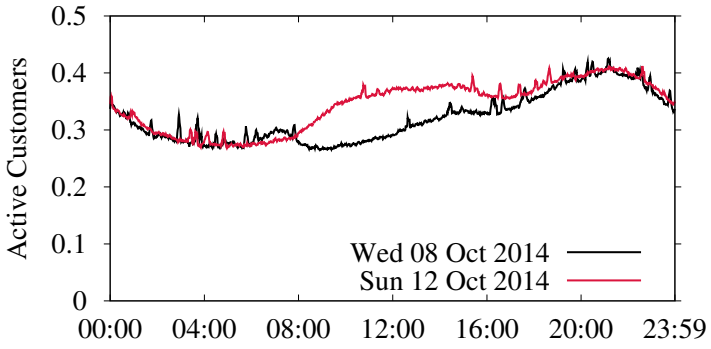


Figure C.4: Fraction of active customers in different days.

served for a period of time T_{out}, after which the resource is returned to the pool of available IP addresses of the CG-NAT.

To conduct our analysis, we first must determine the number of active customers, and observe how their activity varies over the day. The analysis is conducted on the monitored PoP, but can be easily extended to the entire ISP customer population. We focus on the traffic directed to destinations outside the ISP network, ignoring the traffic internal to the ISP, which is not subject to the CG-NAT. We consider TCP and UDP traffic. In particular for TCP, we take into account both successfully completed and failed connections, as in both cases the CG-NAT has to allocate a public IP address and/or a block of port. We suppose that any TCP and UDP connection requires a dedicated (IP address, port) pair on the NAT, and this association must be maintained for the whole connection lifetime. The (IP address, port) pair will be released, freeing the resource, only later T_{out} minutes have passed.

For the experiments in this section we pick a workday (Wednesday) and an off day (Sunday), so to consider different activity patterns.

NAT based on Simple Address Mapping

We first emulate the resource usage in the simple NAT scenario. We expect that in the worst case, i.e., when all customers are active, the ISP would need as many public IP addresses as the overall customer population inside the network. From the NAT perspective, varying T_{out} influences the number of active customers in the network. We suppose T_{out} is 5 min. Fig. C.4 shows the evolution, over 1-day, of the active customers, for both the weekday and the off day. The result shows that a simple NAT policy would turn into considering active approximately 40% of customers that are active at the same time, and hence the ISP would save roughly up to 60% of the public

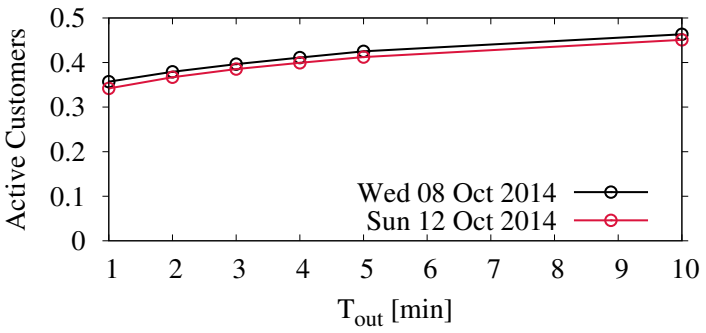


Figure C.5: Maximum fraction of active customers vs NAT T_{out}.

IP addresses. Finally, observe that the user activity is rather regular and similar for different days of the week. We conduct the same experiment on different days, and we observe very similar results (omitted for briefness).

For completeness, we check the impact of T_{out} on the estimation of active customers. In fact, the larger T_{out}, the longer the customer appears as active to the NAT, and the longer the time the NAT has to wait before redeem the public IP address. To this end, we show in Fig. C.5 how the maximum fraction of active customers measured in the day (typically reached at the evening) changes when the T_{out} varies between 1 and 10 min.[2] As shown, the fraction of customers which have to be considered as active increases by a 10% only when increasing T_{out} up to 10 min. Observe also that there is no substantial difference between different weekdays.

CGN based on PAT Policy

While above NATing technique might reduce the pool of public IP addresses to use, the actual savings are still limited, as the number of concurrent active customers is considerably large. Therefore, we investigate the resource requirements when NAT and PAT policy is in place, i.e., each active customer is given a block of ports on a public IP address.

For this policy, it is crucial to dimension the size of the block of ports the CG-NAT shall allocate per customer. Hence, we have to count the per-customer number of concurrent connections. We expect the CG-NAT to assign continuous bulks of ports to each customer. The sizing of the block of

[2]Notice that RFCs suggest to set T_{out} to 2 min for UDP [146] and 2 h for TCP [132]. However, the suggested thresholds have been shown to be too long, and they lead suboptimal retention policies [184]. For this reason, we explore a threshold space closer to the order of tens of minutes.

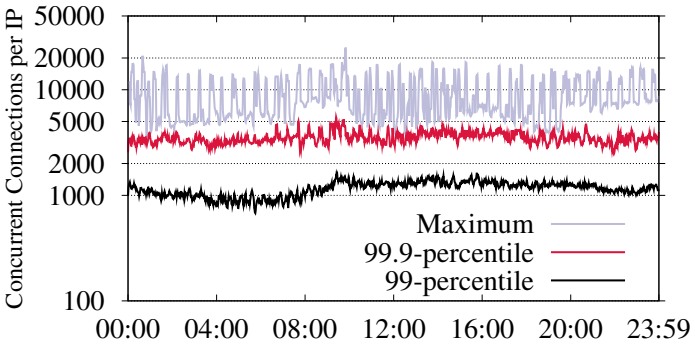


Figure C.6: Maximum, 99.- and 99.9- percentile of the number of per-customer concurrent active connections (computed as the maximum between the numbers of UDP and TCP connections).

ports should be based on the transport protocol, i.e., TCP or UDP, employing the largest number of ports. For instance, let p_{TCP} and p_{UDP} be the number of concurrent active TCP and UDP connections, respectively. The block size must then be larger than $\max(p_{TCP}, p_{UDP})$. We proceed as follows. We choose $T_{out} = 5\,\text{min}$, and we consider as concurrent the connections observed in 1 min long time bin. For each customer, we count the numbers of concurrent TCP and UDP connections. We then pick the maximum between the two and use the result to build a per-minute distributions. In Fig. C.6 we report the maximum, the 99.9- and the 99-percentiles obtained from the per-minute distributions and their evolution over time.

As shown, the number of per-costumer parallel connections rarely overcomes 20,000. In fact, we observe that customers employing a so wide number of ports are mostly users running P2P applications which open many parallel UDP flows. We see that 99% of customers never use more than 2,000 concurrent connections. Allocating a bulk of 2,000 ports for each customer would allow the ISP to use one public IP address for 32 customers. Considering a more conservative approach, i.e., adopting the 99.9-percentile as a reference, we observe roughly 6,000 concurrent connections, leading in this case to allocate about 6 customers per public IP address. For the sake of completeness, we conduct the same experiment picking different days. The results are consistent and lead to very similar conclusions.

Bibliography

[1] P. H. Salus, *Casting the Net: From ARPANET to Internet and Beyond...* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, and K. Cho, "Seven years and one day: Sketching the evolution of internet traffic," in *INFOCOM 2009, IEEE*, pp. 711–719, April 2009.

[3] D. Belson, J. Thompson, M. McKeay, B. Brenner, R. Möller, M. Sintorn, and G. Huston, "Akamai's – state of the internet," Tech. Rep. Q2, Akamai, Inc., 2014.

[4] Cisco Systems, Inc., "Cisco visual networking index: Forecast and methodology, 2014–2019," tech. rep., Cisco Systems, Inc., 2015.

[5] S. Dustdar and W. Schreiner, "A survey on web services composition," *Int. J. Web Grid Serv.*, vol. 1, pp. 1–30, Aug. 2005.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, pp. 50–58, Apr. 2010.

[7] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of content distribution networks," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, (New York, NY, USA), pp. 169–182, ACM, 2001.

[8] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, (New York, NY, USA), pp. 281–287, ACM, 2010.

[9] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, pp. 6:1–6:42, Mar. 2008.

[10] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The Cost of the "S" in HTTPS," in *Proc. ACM CoNEXT*, pp. 133–140, 2014.

[11] K. Claffy, *Internet Traffic Characterization*. PhD thesis, UC San Diego, Jun 1994.

[12] C. Williamson, "Internet traffic measurement," *Internet Computing, IEEE*, vol. 5, pp. 70–74, Nov 2001.

[13] M. Naldi and L. Mastroeni, "Cloud Storage Pricing: A Comparison of Current Practices," in *Proceedings of the HotTopiCS*, pp. 27–34, 2013.

[14] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: A viable solution?," in *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*, DADC '08, (New York, NY, USA), pp. 55–64, ACM, 2008.

[15] I. Drago, M. Mellia, M. M. Munafò, A. Sperotto, R. Sadre, and A. Pras, "Inside Dropbox: Understanding Personal Cloud Storage Services," in *Proceedings of the IMC*, pp. 481–494, 2012.

[16] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The nepenthes platform: An efficient approach to collect malware," in *Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection*, RAID'06, (Berlin, Heidelberg), pp. 165–184, Springer-Verlag, 2006.

[17] J. François, S. Wang, R. State, and T. Engel, "Bottrack: tracking botnets using netflow and pagerank," in *NETWORKING 2011*, pp. 1–14, Springer, 2011.

[18] iMPERVA, "Assessing the effectiveness of antivirus solutions." `http://www.imperva.com/docs/HII_Assessing_the_Effectiveness_of_Antivirus_Solutions.pdf`, 2012.

[19] L. Popa, A. Ghodsi, and I. Stoica, "HTTP as the Narrow Waist of the Future Internet," in *Proc. 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[20] Google Inc., "Spdy: An experimental protocol for a faster web." https://www.chromium.org/spdy/spdy-whitepaper.

[21] M. Belshe, R. Peon, M. Thomson, and A. Melnikov, "Spdy protocol, draft-ietf-httpbis-http2-00," tech. rep., Internet Engineering Task Force, 2013.

[22] Google Inc., "QUIC, a multiplexed stream transport over UDP." https://www.chromium.org/quic.

[23] A. Callado, C. Kamienski, G. Szabo, B. P. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification," *IEEE Communications Surveys Tutorials*, vol. 11, pp. 37–52, rd 2009.

[24] B. Trammel, L. Zheng, S. Silva, and M. Bagnulo, "Hybrid Measurement using IPPM Metrics," tech. rep., Internet Engineering Task Force, Aug. 2014.

[25] The Apache Software Foundation, "Apache Hive TM." https://hive.apache.org/.

[26] B. Claise, "Cisco Systems NetFlow Services Export Version 9," tech. rep., Internet Engineering Task Force, Oct. 2004.

[27] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," tech. rep., Internet Engineering Task Force, May 1990.

[28] M. Mellia, M. Meo, L. Muscariello, and D. Rossi, "Passive Analysis of TCP Anomalies," *Computer Networks*, vol. 52, no. 14, pp. 2663–2676, 2008.

[29] "Tstat." http://tstat.tlc.polito.it/.

[30] M. Trevisan, A. Finamore, M. Mellia, M. Munafò, and D. Rossi, "DPDKStat: 40Gbps Statistical Traffic Analysis with Off-the-Shelf Hardware," tech. rep., Politecnico di Torino, 2016.

[31] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "Kiss: Stochastic packet inspection classifier for udp traffic," *IEEE/ACM Transactions on Networking*, vol. 18, pp. 1505–1515, Oct 2010.

[32] I. N. Bermudez, M. Mellia, M. M. Munafo, R. Keralapura, and A. Nucci, "Dns to the rescue: Discerning content and services in a tangled web," in *Proceedings of the 2012 ACM Conference on Internet*

Measurement Conference, IMC '12, (New York, NY, USA), pp. 413–426, ACM, 2012.

[33] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1, 1999," tech. rep., Internet Engineering Task Force, 2006.

[34] P. Mockapetris, "RFC 1034 - domain names: concepts and facilities (november 1987)," tech. rep., Internet Engineering Task Force, 2003.

[35] R. A. Kemmerer and G. Vigna, "Intrusion detection: A brief history and overview," *Computer*, vol. 35, pp. 27–30, Apr. 2002.

[36] "Snort." `https://www.snort.org/`.

[37] RIPE Network Coordination Center, "Test Traffic Measurement Service." `https://www.ripe.net/analyse/archived-projects/ttm/test-traffic-measurement-service`.

[38] RIPE Network Coordination Center, "DNS Monitoring Service." `https://atlas.ripe.net/dnsmon/`.

[39] Center for Applied Internet Data Analysis (CAIDA), "Archipelago Measurement Infrastructure." `http://www.caida.org/projects/ark/`.

[40] Center for Applied Internet Data Analysis (CAIDA), "Macroscopic Internet Topology Data Kit." `http://www.caida.org/data/internet-topology-data-kit/`.

[41] ThousandEyes Inc. `https://www.thousandeyes.com/`.

[42] AppNeta. `https://www.appneta.com/`.

[43] Anturis Inc. `https://anturis.com/`.

[44] ntop.org. `http://www.ntop.org/products/deep-packet-inspection/ndpi/`.

[45] Endace Technology Ltd. `https://www.endace.com/`.

[46] A. Smola and S. Vishwanathan, eds., *Introduction to Machine Learning*. Cambridge, UK: Cambridge University Press, Inc., 2008.

[47] W. Klösgen and J. M. Zytkow, eds., *Handbook of Data Mining and Knowledge Discovery*. New York, NY, USA: Oxford University Press, Inc., 2002.

[48] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 8, pp. 1226–1238, 2005.

[49] L. Eadicicco, "Google Drive's New Pricing Plans Blow Dropbox Away," 2014. http://www.businessinsider.com/google-drive-pricing-2014-3.

[50] W. Hu, T. Yang, and J. N. Matthews, "The Good, the Bad and the Ugly of Consumer Cloud Storage," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 110–115, 2010.

[51] H. Wang, R. Shea, F. Wang, and J. Liu, "On the Impact of Virtualization on Dropbox-Like Cloud File Storage/Synchronization Services," in *Proceedings of the IWQoS*, pp. 11:1–11:9, 2012.

[52] T. Mager, E. Biersack, and P. Michiardi, "A Measurement Study of the Wuala On-line Storage Service," in *Proceedings of the P2P*, pp. 237–248, 2012.

[53] G. Gonçalves, I. Drago, A. P. C. da Silva, A. B. Vieira, and J. M. de Almeida, "Modeling the Dropbox Client Behavior," in *Proceedings of the ICC*, pp. 1332–1337, 2014.

[54] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, "Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space," in *Proceedings of the SEC*, pp. 1–11, 2011.

[55] D. Kholia and P. Wegrzyn, "Looking Inside the (Drop) Box," in *Proceedings of the WOOT*, 2013.

[56] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proceedings of the IMC*, pp. 1–14, 2010.

[57] A. Bergen, Y. Coady, and R. McGeer, "Client Bandwidth: The Forgotten Metric of Online Storage Providers," in *Proceedings of the PacRim*, pp. 543–548, 2011.

[58] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, and Z.-L. Zhang, "Towards Network-Level Efficiency for Cloud Storage Services," in *Proceedings of the IMC*, 2014.

[59] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai, "Efficient Batched Synchronization in Dropbox-Like Cloud Storage Services," in *Proceedings of the Middleware*, pp. 307–327, 2013.

[60] R. Gracia-Tinedo, M. S. Artigas, A. Moreno-Martinez, C. Cotes, and P. G. Lopez, "Actively Measuring Personal Cloud Storage," in *Proceedings of the CLOUD*, pp. 301–308, 2013.

[61] S. Liu, X. Huang, H. Fu, and G. Yang, "Understanding Data Characteristics and Access Patterns in a Cloud Storage System," in *Proceedings of the CCGrid*, pp. 327–334, 2013.

[62] P. Casas and R. Schatz, "Quality of Experience in Cloud Services: Survey and Measurements," *Comput. Netw.*, vol. 68, pp. 149–165, 2014.

[63] P. Amrehn, K. Vandenbroucke, T. Hossfeld, K. D. Moor, M. Hirth, R. Schatz, and P. Casas, "Need for Speed? On Quality of Experience for Cloud-based File Storage Services," in *Proceedings of the PQS*, pp. 184–190, 2013.

[64] Y. Zhang, C. Dragga, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "ViewBox: Integrating Local File Systems with Cloud Storage Services," in *Proceedings of the FAST*, pp. 119–132, 2014.

[65] I. Drago, *Understanding and Monitoring Cloud Services*. PhD thesis, University of Twente, 2013.

[66] I. N. Bermudez, S. Traverso, M. Mellia, and M. M. Munafò, "Exploring the Cloud from Passive Measurements: The Amazon AWS Case," in *INFOCOM*, pp. 230–234, 2013.

[67] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the Expansion of Google's Serving Infrastructure," in *Proceedings of the IMC*, pp. 313–326, 2013.

[68] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye, "IP Geolocation Databases: Unreliable?," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, pp. 53–56, 2011.

[69] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafò, and S. Rao, "Dissecting Video Server Selection Strategies in the YouTube CDN," in *Proceedings of the ICDCS*, pp. 248–257, 2011.

[70] B. Eriksson and M. Crovella, "Understanding Geolocation Accuracy using Network Geometry," in *INFOCOM*, pp. 75–79, 2013.

[71] A. Tridgell, *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, 1999.

[72] R. Sadre and B. R. Haverkort, "Fitting Heavy-Tailed HTTP Traces with the New Stratified EM-Algorithm," in *Proceedings of the IT-NEWS*, pp. 254–261, 2008.

[73] E. Bocchi and I. Drago, "Cloud benchmarks." `https://www.simpleweb.org/wiki/index.php/Cloud_benchmarks`.

[74] Kaspersky Lab, "Global corporate it security risks: 2013." `http://media.kaspersky.com/en/business-security/Kaspersky_Global_IT_Security_Risks_Survey_report_Eng_final.pdf`, 2013.

[75] Symantec, "2014 internet security threat report." `http://www.symantec.com/security_response/publications/threatreport.jsp`, 2014.

[76] J. Nazario, "Phoneyc: A virtual client honeypot," in *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'09, (Berkeley, CA, USA), pp. 6–6, USENIX Association, 2009.

[77] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, "Rozzle: De-cloaking internet malware," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, (Washington, DC, USA), pp. 443–457, IEEE Computer Society, 2012.

[78] J. Zhang, P. Porras, and J. Ullrich, "Highly predictive blacklisting," in *Proceedings of the 17th Conference on Security Symposium*, SS'08, (Berkeley, CA, USA), pp. 107–122, USENIX Association, 2008.

[79] B. Krebs, "Antivirus is dead: Long live antivirus!." `http://krebsonsecurity.com/2014/05/antivirus-is-dead-long-live-antivirus/`, 2014.

[80] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 39–53, Apr. 2004.

[81] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pp. 41–52, ACM, 2006.

[82] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the 2007 ACM Workshop on Recurring Malcode*, WORM '07, 2007.

[83] L. Zhang and Y. Guan, "Detecting click fraud in pay-per-click streams of online advertising networks," in *Distributed Computing Systems, 2008. ICDCS '08. The 28th International Conference on*, pp. 77–84, June 2008.

[84] N. Kshetri, "The economics of click fraud," *IEEE Security & Privacy*, vol. 8, pp. 45–53, May 2010.

[85] C. Grier *et al.*, "Manufacturing compromise: The emergence of exploit-as-a-service," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, (New York, NY, USA), pp. 821–832, ACM, 2012.

[86] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, (New York, NY, USA), pp. 281–290, ACM, 2010.

[87] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting malicious flux service networks through passive analysis of recursive DNS traces," in *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, pp. 311–320, 2009.

[88] N. Jiang, J. Cao, Y. Jin, L. Li, and Z.-L. Zhang, "Identifying Suspicious Activities Through DNS Failure Graph Analysis," in *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pp. 144–153, IEEE, 2010.

[89] Y. Nadji, M. Antonakakis, R. Perdisci, and W. Lee, "Connected colors: Unveiling the structure of criminal networks," in *Research in Attacks, Intrusions, and Defenses*, pp. 390–410, Springer, 2013.

[90] P. K. Manadhata, S. Yadav, P. Rao, and W. Horne, "Detecting malicious domains via graph inference," in *ESORICS 2014*, pp. 1–18, Springer, 2014.

[91] L. Liu, S. Saha, R. Torres, J. Xu, P.-N. Tan, A. Nucci, and M. Mellia, "Detecting Malicious Clients in ISP Networks Using HTTP Connectivity Graph and Flow Information," in *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pp. 150–157, IEEE, 2014.

[92] L. Invernizzi, S. Miskovic, R. Torres, S. Saha, S.-J. Lee, C. Kruegel, and G. Vigna, "Nazca: Detecting Malware Distribution in Large-Scale Networks," in *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS '14)*, Feb 2014.

[93] A. Le, A. Markopoulou, and M. Faloutsos, "PhishDef: URL names say it all," in *Proc. of the 30th IEEE Int'l Conference on Computer Communications*, pp. 191–195, 2011.

[94] A. Oza, K. Ross, R. M. Low, and M. Stamp, "Http attack detection using n-gram analysis," *Elsevier Computers & Security*, vol. 45, pp. 242–254, 2014.

[95] M. Antonakakis, R. Perdisci, Y. Nadji, N. V. II, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of DGA-based malware," in *Proc. of USENIX Security Symposium*, pp. 491–506, 2012.

[96] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th USENIX Conference on Security Symposium*, SS'08, pp. 139–154, 2008.

[97] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: detecting malware infection through IDS-driven dialog correlation," in *Proc. of the 16th USENIX Security Symposium*, pp. 12:1–12:16, 2007.

[98] C. J. Dietrich, C. Rossow, and N. Pohlmann, "Cocospot: Clustering and recognizing botnet command and control channels using traffic analysis," *Computer Networks*, vol. 57, no. 2, pp. 475–486, 2013.

[99] H. Hang, X. Wei, M. Faloutsos, and T. Eliassi-Rad, "Entelecheia: Detecting P2P botnets in their waiting stage," in *IFIP Networking Conference, 2013*, pp. 1–9, 2013.

[100] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious URLs," in *Proc. of the ACM SIGKDD*, pp. 1245–1254, 2009.

[101] J. Zhang, C. Seifert, J. W. Stokes, and W. Lee, "Arrow: Generating signatures to detect drive-by downloads," in *Proc. of the 20th Int'l Conference on World Wide Web*, pp. 187–196, 2011.

[102] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proc. of the Network and Distributed System Security Symposium*, 2008.

[103] M. I. o. T. Lincoln Laboratory, "Darpa intrusion detection data sets." `https://www.ll.mit.edu/ideval/data/`, 1998.

[104] R. Perdisci, "Mcpad – attack dataset." `http://roberto.perdisci.com/projects/mcpad`, 2009.

[105] K. L. Ingham and H. Inoue, "Comparing anomaly detection techniques for http," in *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, RAID'07, (Berlin, Heidelberg), pp. 42–62, Springer-Verlag, 2007.

[106] A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna, "Escape from monkey island: Evading high-interaction honeyclients," in *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'11, (Berlin, Heidelberg), pp. 124–143, Springer-Verlag, 2011.

[107] P. Porras, "Inside risks: Reflections on conficker," *Communications of ACM*, vol. 52, Oct. 2009.

[108] L. Seltzer, "Conficker: Still spamming after all these years." `http://www.zdnet.com/conficker-still-spamming-after-all-these-years-7000031206/`, 2014.

[109] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2nd ed., 2013.

[110] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma, "Discovering all most specific sentences," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 2, pp. 140–174, 2003.

[111] F. Pan, G. Cong, A. K. Tung, J. Yang, and M. J. Zaki, "Carpenter: Finding closed patterns in long biological datasets," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 637–642, ACM, 2003.

[112] "VirusTotal." `https://www.virustotal.com/`.

[113] D. Thaler, "Evolution of the ip model," tech. rep., Internet Engineering Task Force, May 2011.

[114] C. Morgan, "IAB Statement on Internet Confidentiality." https://www.iab.org/2014/11/14/iab-statement-on-internet-confidentiality, Nov 2014.

[115] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol, version 2 – HTTP/2, 2015," tech. rep., Internet Engineering Task Force, May 2015.

[116] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is the web http/2 yet?," in *International Conference on Passive and Active Network Measurement*, pp. 218–232, Springer, 2016.

[117] Amazon Inc. http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales.

[118] S. Souders, "Velocity and the bottom line." http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html.

[119] S. Jiang, D. Guo, and B. Carpenter, "RFC 6264 - An Incremental Carrier-Grade NAT (CGN) for IPv6 Transition," tech. rep., Internet Engineering Task Force, 2011.

[120] Titus, Tobin and Mann, Jatinder and Jain, Arvind , "Navigation Timing Level 2, W3C Editor's Draft 22." http://w3c.github.io/navigation-timing/, Apr. 2016.

[121] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDY'Ier Mobile Web?," in *Proc. ACM CoNEXT*, pp. 303–314, December 2013.

[122] F. Qian, V. Gopalakrishnan, E. Halepovic, S. Sen, and O. Spatscheck, "TM3: Flexible Transport-layer Multi-pipe Multiplexing Middlebox Without Head-of-line Blocking," in *Proc. ACM CoNEXT*, December 2015.

[123] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?," in *Proc. USENIX NSDI*, pp. 387–399, Apr. 2014.

[124] X. S. Wang, A. Krishnamurthy, and D. Wetherall, "Speeding up Web Page Loads with Shandian," in *Proc. USENIX NSDI*, pp. 109–122, Mar. 2016.

[125] Alexa Internet Inc. http://www.alexa.com.

[126] Google Inc. https://googlewebmastercentral.blogspot.fr/2010/04/using-site-speed-in-web-search-ranking.html.

[127] Google Inc. http://googleresearch.blogspot.fr/2009/06/speed-matters.html.

[128] Google Inc. https://www.youtube.com/watch?v=muSIzHurn4U.

[129] J. Brutlag, Z. Abrams, and P. Meenan, "Above the fold time: Measuring web page performance visually." http://conferences.oreilly.com/velocity/velocity-mar2011/public/schedule/detail/18692, Mar. 2011.

[130] Google Inc. https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index.

[131] I. Grigorik, *High Performance Browser Networking*. O'Reilly Media, 2013.

[132] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh, "RFC 5382 - NAT Behavioral Requirements for TCP," tech. rep., Internet Engineering Task Force, 2008.

[133] S. Perreault, I. Yamagata, S. Miyakawa, A. Nakagawa, and H. Ashida, "RFC 6888 - Common Requirements for Carrier-Grade NATs (CGNs)," tech. rep., Internet Engineering Task Force, 2013.

[134] P. Srisuresh, B. Ford, and D. Kegel, "RFC 5128 - State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)," tech. rep., Internet Engineering Task Force, 2008.

[135] J. Rosenberg, A. Keranen, B. B. Lowekamp, and A. B. Roach, "RFC 6544 - TCP Candidates with Interactive Connectivity Establishment (ICE)," tech. rep., Internet Engineering Task Force, 2012.

[136] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar, "Klotski: Reprioritizing Web Content to Improve User Experience on Mobile Devices," in *Proc. USENIX NSDI*, pp. 439–453, May 2015.

[137] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate Record-and-Replay for HTTP," in *Proc. USENIX ATC*, pp. 417–429, July 2015.

[138] H. de Saxcé, I. Oprescu, and Y. Chen, "Is http/2 really faster than http/1.1?," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 293–299, April 2015.

[139] K. Zarifis, M. Holland, M. Jain, E. Katz-Bassett, and R. Govindan, "Modeling HTTP/2 Speed from HTTP/1 Traces," in *Proc. Passive and Active Measurement (PAM)*, 2016.

[140] M. Varvello, D. Blackburn, Jeremy abd Naylor, and K. Papagiannaki, "eyeorg – A platform for crowdsourcing Web quality of experience measurements." `https://eyeorg.net/`.

[141] Google Inc. `https://www.ampproject.org/`.

[142] S. Khirman and P. Henriksen, "Relationship between quality-of-service and quality-of-experience for public internet service," in *In Proc. of the 3rd Workshop on Passive and Active Measurement*, 2002.

[143] International Telecommunication Union – ITU, "Recommendation G.1030: Estimating end-to-end performance in IP networks for data applications," tech. rep., International Telecommunication Union – ITU, Feb. 2005.

[144] International Telecommunication Union – ITU, "Methods for the subjective assessment of video quality, audio quality and audiovisual quality of internet video and distribution quality television in any environment." ITU-T Recommendation P.913, Sept. 2014.

[145] International Telecommunication Union – ITU, "Subjective testing methodology for web browsing," tech. rep., International Telecommunication Union – ITU, Feb. 2014.

[146] F. Audet, and C. Jennings, "RFC 4787 - Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," tech. rep., Internet Engineering Task Force, 2007.

[147] L. DiCioccio, R. Teixeira, M. May, and C. Kreibich, "Probe and pray: Using upnp for home network measurements," in *Proceedings of the 13th International Conference on Passive and Active Measurement*, PAM'12, (Berlin, Heidelberg), pp. 96–105, Springer-Verlag, 2012.

[148] S. M. Bellovin, "A Technique for Counting Natted Hosts," in *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, (New York, NY, USA), pp. 267–272, ACM, 2002.

[149] G. Maier, F. Schneider, and A. Feldmann, "Nat usage in residential broadband networks," in *Proceedings of the 12th International Conference on Passive and Active Measurement*, PAM'11, (Berlin, Heidelberg), pp. 32–41, Springer-Verlag, 2011.

[150] V. Krmicek, J. Vykopal, and R. Krejci, "Netflow Based System for NAT Detection," in *Proceedings of the 5th International Student Workshop on Emerging Networking Experiments and Technologies*, Co-Next Student Workshop '09, (New York, NY, USA), pp. 23–24, ACM, 2009.

[151] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang, "An Untold Story of Middleboxes in Cellular Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (New York, NY, USA), pp. 374–385, ACM, 2011.

[152] N. Škoberne, O. Maennel, I. Phillips, R. Bush, J. Zorz, and M. Ciglaric, "IPv4 Address Sharing Mechanism Classification and Tradeoff Analysis," *IEEE/ACM Trans. Netw.*, vol. 22, pp. 391–404, Apr. 2014.

[153] Y. Ohara, K. Nishizuka, K. Chinen, K. Akashi, M. Kohrin, E. Muramoto, and S. Miyakawa, "On the Impact of Mobile Network Delays on Connection Establishment Performance of a Carrier Grade NAT Device," in *Proceedings of the AINTEC 2014 on Asian Internet Engineering Conference*, AINTEC '14, (New York, NY, USA), pp. 1:1–1:8, ACM, 2014.

[154] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, (Berkeley, CA, USA), pp. 13–13, USENIX Association, 2005.

[155] C. Donley, L. Howard, V. Kuarsingh, J. Berg, and J. Doshi, "RFC 7021 - Assessing the Impact of Carrier-Grade NAT on Network Applications," tech. rep., Internet Engineering Task Force, 2013.

[156] "Yslow ruleset matrix." `http://yslow.org/ruleset-matrix/`.

[157] "Pagespeed insights." `https://developers.google.com/speed/docs/insights/rules`.

[158] Dynatrace, "dynatrace." `http://dynatrace.com/`.

[159] Google Inc. `http://webpagetest.org`.

[160] http://www.showslow.com.

[161] World Wide Web Consortium. https://www.w3.org/TR/2012/REC-navigation-timing-20121217.

[162] M. Bailey, D. Dittrich, E. Kenneally, and D. Maughan, "The menlo report," *IEEE Security & Privacy*, vol. 10, no. 2, pp. 71–75, 2012.

[163] M. Alllman and V. Paxson, "Issues and etiquette concerning use of shared measurement data," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, (New York, NY, USA), pp. 135–140, ACM, 2007.

[164] Apache HTTP Server Project. https://httpd.apache.org/docs/2.4/new_features_2_4.html.

[165] K. B. Ariyur and M. Krstic, *Real-time optimization by extremum-seeking control*. John Wiley & Sons, 2003.

[166] Google Inc. https://developer.chrome.com/devtools.

[167] Google Inc. https://github.com/cyrus-and/chrome-har-capturer.

[168] I. Grigorik, "Http/2 is here, let's optimize!." http://bit.ly/http2-opt.

[169] R. Peon and H. Ruellan, "HPACK: Header Compression for HTTP/2," tech. rep., Internet Engineering Task Force, May 2015.

[170] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), (New York, NY, USA), pp. 267–277, ACM, 1968.

[171] J. Nielsen, "Response times: The 3 important limits." https://www.nngroup.com/articles/response-times-3-important-limits/, Jan. 1993.

[172] P. Reichl, S. Egger, R. Schatz, and A. D'Alconzo, "The Logarithmic Nature of QoE and the Role of the Weber-Fechner Law in QoE Assessment," in *Proc. IEEE ICC*, pp. 1–5, May 2010.

[173] P. Irish, "Delivering the goods in under 1000ms." http://bit.ly/1toUUA7.

[174] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with WProf," in *Proc. UNIX NSDI*, pp. 473–485, 2013.

[175] C. Donley, L. Howard, V. Kuarsingh, A. Chandrasekaran, and V. Ganti, "Assessing the Impact of NAT444 on Network Applications," tech. rep., Internet Engineering Task Force (IETF), 2011.

[176] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing Skype Traffic: When Randomness Plays with You," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 37–48, Aug. 2007.

[177] R. Birke, M. Mellia, M. Petracca, and D. Rossi, "Understanding VoIP from backbone measurements," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 2027–2035, IEEE, 2007.

[178] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi, "Detailed Analysis of Skype Traffic," *Multimedia, IEEE Transactions on*, vol. 11, pp. 117–127, Jan 2009.

[179] A. L. Gibbs and F. E. Su, "On Choosing and Bounding Probability Metrics," *International statistical review*, vol. 70, no. 3, pp. 419–435, 2002.

[180] A. Trifilò, S. Burschka, and E. Biersack, "Traffic to protocol reverse engineering," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1–8, July 2009.

[181] Eyeo GmbH, "AdBlock Plus." `https://adblockplus.org/`.

[182] Z. Hu, "NAT traversal techniques and peer-to-peer applications," in *HUT T-110.551 Seminar on Internetworking*, pp. 04–26, 2005.

[183] J. Rosenberg, R. Mahy, P. Matthews and D. Wing, "RFC 5389 - Session Traversal Utilities for NAT (STUN)," tech. rep., Internet Engineering Task Force, 2008.

[184] S. Alcock, R. Nelson, and M. David, "Investigating the Impact of Service Provider NAT on Residential Broadband Users," tech. rep., University of Waikato, 2010.

Acronyms

ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
ATF	Above-the-fold
BI	ByteIndex
C&C	Command & Control
CCDF	Complementary CDF
CDF	Cumulative Distribution Function
CDN	Content Delivery Network
CG	Connectivity Graph
CG-NAT	Carrier Grade NAT
CHR	Google Chrome
CSS	Cascading Style Sheets
DDoS	Distributed Denial of Service
DGA	Domain Generated Algorithm
DNS	Domain Name System
DOM	Document Object Model
DPI	Deep Packet Inspection
EK	Exploit Kit
FN	False Negative
FP	False Positive
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
FTTH	Fiber to the Home
GUI	Graphical User Interface
H-IDS	Host IDS
H1	HTTP/1.*

H2	HTTP/2
HAR	HTTP ARchive
Host-CG	Host Connectivity Graph
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IPS	Intrusion Prevention System
IRC	Internet Relay Chat
ISP	Internet Service Provider
IXP	Internet Exchange Point
JS	Jensen-Shannon Divergence
KPI	Key Performance Indicator
L4	Transport Layer
L7	Application Layer
LAN	Local Area Network
MOS	Mean Opinion Score
mRMR	Minimum Redundancy, Maximum Relevance
N-IDS	Network IDS
NAT	Network Address Translation
OI	ObjectIndex
OS	Operating System
P2P	Peer-To-Peer
PAT	Port Address Translation
PDF	Probability Distribution Function
PLT	Page Load Time
PoP	Point of Presence
POP3	Post Office Protocol
PPV	Positive Predicted Value
PUP	Potentially Unwanted Program
QoE	Quality of Experience
QoS	Quality of Service

QUIC	Quick UDP Internet Connections
RF	Random Forest
RFC	Request For Comment
RPC	Remote Procedure Call
RTT	Round Trip Time
SDM	Statistical Distance Measure
Seed-CG	Seed Connectivity Graph
SI	SpeedIndex
SMTP	Simple Mail Transfer Protocol
SPDY	SPDY - An experimental protocol for a faster web
SSL	Secure Sockets Layer
TCP	Transfer Control Protocol
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
TPR	True Positive Rate
Tstat	TCP STatistic and Analysis Tool
TTFB	Time to the First Byte
TTFP	Time to the First Paint
TTL	Time to Live
TTLB	Time to the Last Byte
TTLP	Time to the Last Paint
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WebQoE	Web Quality of Experience
WPT	WebPagetest

About the Author

I was born in Torino, Italy, on September 28th, 1989. I received my Master of Science (M.Sc.) degree and my Bachelor (B.Sc.) degree in Telecommunications Engineering from Politecnico di Torino in 2013 and 2011, respectively. From 2014 until 2016, I was a Ph.D. student enrolled in a Joint-Ph.D. program between Politecnico di Torino, Italy, and Télécom ParisTech, France, working at the Telecommunication Networks Group (TNG) under the supervision of Prof. Marco Mellia and at the Laboratory of Information, Networking and Communication Science (LINCS) under the supervision of Prof. Dario Rossi, respectively. During my first year as a Ph.D. student, I had the opportunity to develop my research activities in cooperation with Narus Inc. (Sunnyvale, CA, USA), now part of Symantec. In summer 2014, I interned at Narus Inc. where I had the pleasure to work with Dr. Sabyasachi Saha and Prof. Sung-Ju Lee.

List of Publications

In the following, the list of papers I published during my research activity:

1. E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the Quality of Experience of Web Users," in *ACM Computer Communication Review*, ACM CCR, Accepted for publication, 2016.

2. A. Morichetta, E. Bocchi, H. Metwalley, and M. Mellia, "CLUE: Clustering for Mining Web URLs," in *International Teletraffic Congress*, ITC 28, Sept 2016.

3. E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the Quality of Experience of Web Users," in *Proceedings of the 2016 Workshop on QoE-based*

Analysis and Management of Data Communication Networks, ACM SIGCOMM Workshops – Internet-QoE '16, pp. 37–42, Aug 2016.

4. E. Bocchi, A. S. Khatouni, S. Traverso, A. Finamore, M. Munafò, M. Mellia, and D. Rossi, "Statistical Network Monitoring: Methodology and Application to Carrier-Grade NAT," in *Elsevier Computer Networks – Special Issue on Machine learning, data mining and Big Data frameworks for network monitoring and troubleshooting*, vol. 107, Part 1, pp. 20–35, 2016.

5. D. Rossi, E. Bocchi, and L. D. Cicco, "Web QoE: Moving beyond Google's SpeedIndex," in *IEEE International Conference on Computer Communications – Innovation Challenge Pitchfest*, IEEE INFOCOM 2016, Apr 2016.
   Top-5 finalist at IEEE Infocom 2016 Innovation Challenge.

6. E. Bocchi, L. Grimaudo, M. Mellia, E. Baralis, S. Saha, S. Miskovic, G. Modelo-Howard, and S.-J. Lee, "MAGMA: Network Behavior Classifier for Malware Traffic," in *Elsevier Computer Networks – Special Issue on Traffic and Performance in the Big Data Era*, vol. 109, Part 2, pp. 142–156, 2016.

7. E. Bocchi, I. Drago, and M. Mellia, "Experiences of Cloud Storage Service Monitoring: Performance Assessment and Comparison," in *Cloud Services for Synchronization and Sharing (CS3)*, pp. 13, Jan 2016.

8. E. Bocchi, I. Drago, and M. Mellia, "Personal Cloud Storage: Usage, Performance and Impact of Terminals," in *IEEE 4th International Conference on Cloud Networking*, IEEE CloudNet 2015, pp. 106–111, Oct 2015.

9. E. Bocchi, A. Khatouni, S. Traverso, A. Finamore, V. Di Gennaro, M. Mellia, M. Munafò, and D. Rossi, "Impact of Carrier-Grade NAT on Web Browsing," in *International Wireless Communications and Mobile Computing Conference*, IWCMC 2015, pp. 532–537, Aug 2015.
   Best Paper Award of TRaffic Analysis and Characterization (TRAC) 2015.

10. E. Bocchi, L. Grimaudo, M. Mellia, E. Baralis, S. Saha, S. Miskovic, G. Modelo-Howard, and S. J. Lee, "Network Connectivity Graph for Malicious Traffic Dissection," in *24th International Conference on Computer Communication and Networks*, ICCCN 2015, pp. 1–9, Aug 2015.

11. E. Bocchi, I. Drago, and M. Mellia, "Personal Cloud Storage Benchmarks and Comparison," in *IEEE Transactions on Cloud Computing*, IEEE TCC, Accepted for publication, 2015.

12. A. Finamore, S. Saha, G. Modelo-Howard, S.-J. Lee, E. Bocchi, L. Grimaudo, M. Mellia, and E. Baralis, "Macroscopic View of Malware in Home Networks," in *IEEE 12th Annual Consumer Communications and Networking Conference*, IEEE CCNC 2015, pp. 262–266, Jan 2015.

13. E. Bocchi, M. Mellia, and S. Sarni, "Cloud Storage Service Benchmarking: Methodologies and Experimentations," in *IEEE 3rd International Conference on Cloud Networking*, IEEE CloudNet 2014, pp. 395–400, Oct 2014.

14. I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, "Benchmarking Personal Cloud Storage," in *Proceedings of the 2013 Conference on Internet Measurement*, ACM IMC '13, pp. 205–212, Oct 2013.