# understanding and monitoring cloud services

idilio drago

# Understanding and Monitoring Cloud Services

Idilio Drago

**Graduation Committee:**

| | |
|---|---|
| Chairman: | Prof. dr. ir. Anton J. Mouthaan |
| Promoter: | Prof. dr. ir. Boudewijn R. Haverkort |
| Assistant promoter: | Dr. ir. Aiko Pras |

| | |
|---|---|
| Prof. dr. Marco Mellia | Politecnico di Torino, Italy |
| Dr. Ramin Sadre | Aalborg University, Denmark |
| Prof. dr. ir. Filip De Turck | Ghent University, Belgium |
| Prof. dr. Jürgen Schönwälder | Jacobs University Bremen, Germany |
| Prof. dr. ing. Paul J. M. Havinga | University of Twente, The Netherlands |
| Prof. dr. Hans van den Berg | University of Twente, The Netherlands |
| Prof. dr. Jos van Hillegersberg | University of Twente, The Netherlands |

CTIT Ph.D. thesis Series No. 13-279
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE
Enschede, the Netherlands

ISBN: 978-90-365-3577-9
ISSN: 1381-3617 (CTIT Ph.D. thesis Series No. 13-279)
DOI: 10.3990/1.9789036535779
http://dx.doi.org/10.3990/1.9789036535779

Typeset with LaTeX. Printed by Ipskamp Drukkers B.V.

# UNDERSTANDING AND MONITORING CLOUD SERVICES

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties,
in het openbaar te verdedigen
op vrijdag 13 december 2013 om 14.45 uur

door

Idilio Drago

geboren op 30 maart 1980
te Colatina-ES, Brazilië

Dit proefschrift is goedgekeurd door:
Prof. dr. ir. Boudewijn R. Haverkort (promotor)
Dr. ir. Aiko Pras (assistent-promotor)

# Acknowledgments

First of all, I would like to sincerely thank my supervisor, very soon a new Professor at the University of Twente, Dr.ir. Aiko Pras and my promoter Prof.dr.ir Boudewijn R. Haverkort. Since our first contact, when I was invited for an interview in Enschede, until the very last comments on the Appendix, it was an enormous pleasure to work with both of them. This thesis would not be possible without their continuous support and guidance.

I would also like to thank the members of my graduation committee, for accepting the invitation to join the committee, and for their effort in reviewing the thesis.

I would like to thank all co-authors of papers I wrote, or helped to write, during my Ph.D. Fortunately, the research of a Ph.D. student is by no means the work of a single person. The collaboration of such a distinguished group of researchers not only helped me to achieve the outcomes presented in this thesis, but also resulted in a strong admiration and friendships. I also express my gratitude to all students that I advised, formally or informally. Working with students is certainly the greatest joy in academia. Thank you all!

Special thanks go to my colleagues at DACS. Working at DACS was always gratifying thanks to the people that form the group. Finally, I am extremely thankful for the encouragement I found in my family and friends, both those I left behind in Brazil and those I met during my time in the Netherlands. Their warmth and closeness, despite the physical distance in some cases, were undoubtedly the best fuel to kept me going on.

This research would not be possible without the financial support received from several sources. Firstly, I would like to thank the Dutch Ministry of Economic Affairs, Agriculture and Innovation for the support via its agency Agentschap NL and its IOP GenCom project *SeQual*. Secondly, this work has been partly funded by the Network of Excellence project *Flamingo* (ICT-318488) and the EU-IP project *mPlane* (n-318627). Both projects are supported by the European Commission under its Seventh Framework Programme. Finally, I would like to thank the European Commission for the grant received in the context of the TMA COST Action IC0703 for a short-term research mission in the *Politecnico di Torino* in Italy in 2011/2012.

# Abstract

Cloud services have changed the way computing power is delivered to customers, by offering computing and storage capacity in remote data centers on demand. The advantages of the cloud model have fast resulted in powerful international providers. However, this success has not come without problems. Cloud providers have repeatedly been related to reports of major failures, including outages, performance degradation and loss of users' data. Similarly, privacy of cloud services is of huge concern for exposing users to providers and, more alarmingly, to foreign governments. The alleged existence of national surveillance programs that rely on information collected from cloud providers indicates privacy threats are real, and they put in check the advantages of using cloud services.

We argue that these issues will drive the developments around cloud services in two directions. Firstly, dependability concerns will impel *enterprise customers* to require assurances and independent monitoring of their services in the cloud. Secondly, privacy issues will prompt *new players* to offer services that combine the strengths of cloud computing with both stronger privacy and protection against foreign governments. Indeed, initial signs of both trends can already be mentioned, such as companies starting to offer independent tools to monitor cloud performance and regional players entering the cloud market while worldwide firms hesitate to trust international providers.

This thesis has two objectives. Firstly, we investigate simple and scalable methods for monitoring the performance of cloud services from the users' point of view, aiming to provide means for customers to monitor services in the cloud easily and independently. Secondly, we study how cloud services are implemented and the implications of their design and usage for the Internet, aiming to foster the development of new services. We focus primarily on *cloud storage*, because, as we will show, it is a popular application already accounting for a significant share of Internet traffic.

Our main contributions are the following: (i) we introduce a novel method to monitor the performance of cloud services, which relies on flow measurements collected from network vantage points without providers' interference; (ii) we apply our method and present the first in-depth characterization of cloud storage, revealing its typical usage and possible performance bottlenecks; and (iii) we

evaluate the implications of design choices for both users and the Internet, by comparing different providers in a series of benchmarks.

Our analyses show that cloud services can be monitored *from outside*, using information normally collected from customers' networks. Our results make clear that cloud storage is data-intensive and understanding its usage is essential for building well-performing services that wisely use the Internet. Moreover, our comparisons of providers demonstrate that design differences that seem minor at first can result in surprisingly costs and serious performance bottlenecks.

Our contributions are valuable for companies outsourcing to the cloud as well as for engineers developing solutions and provisioning resources for cloud storage. Overall, our analyses, algorithms and datasets are a great asset to anticipate the impact of a massive adoption of such services, and can assist *private* and *national* cloud providers to develop a next generation of well-performing cloud storage services.

# Contents

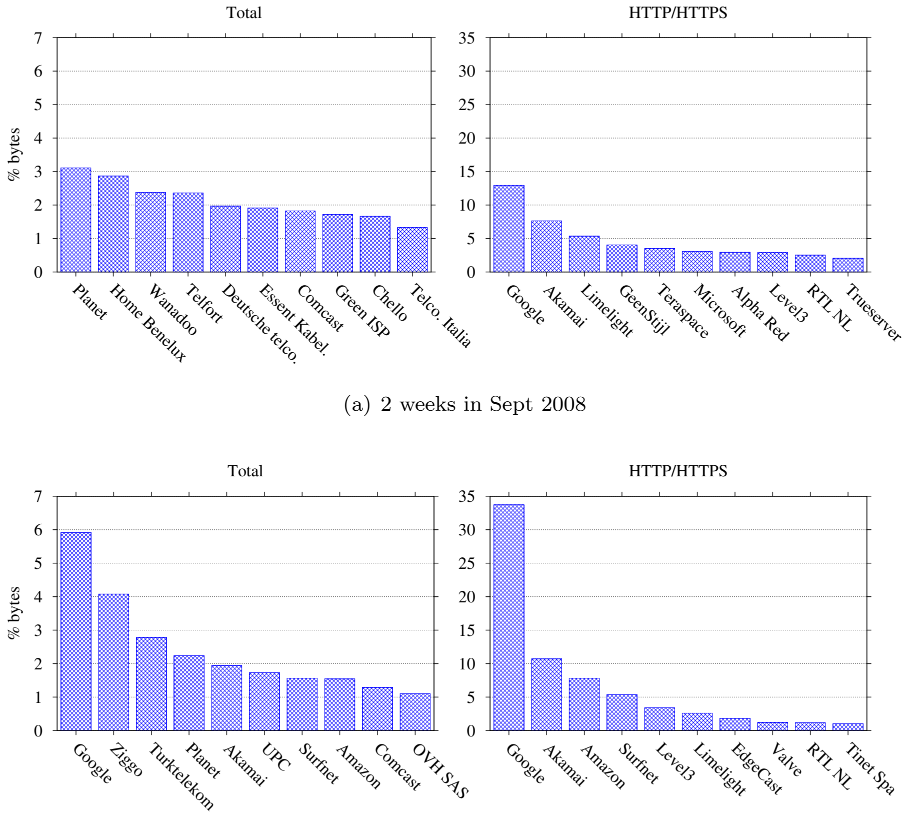
CHAPTER 1

# Introduction

Cloud services have changed the way computing power is delivered to customers. Cloud services abstract away the complexity of system management, by offering computing and storage capacity in remote data centers on demand. In retrospective, this advent can be seen as a natural step in the evolution of the Internet [9]. The extreme growth of Web services popularity in the early 2000's led providers, such as Amazon, Google and Microsoft, to invest both in data center provisioning for their own services and in the development of scalable software solutions [9]. Even though the later conversion of this infrastructure into a utility may have involved major technical challenges, the way for a new computing model was certainly starting to be paved.

The success of this new model can be demonstrated by the increasing traffic to the biggest cloud providers. Labovitz et al. [94] – in a measurement study covering around 25 % of the Internet inter-domain traffic between 2007 and 2009 – showed that a very small number of networks is involved in most Internet traffic. Among more than 30,000 Autonomous Systems (ASs), only 30 are responsible for around 30 % of all inter-domain exchanges. The top 150 ASs are already involved in more than 50 % of the transfers. Major cloud providers are topping the list, with Google being responsible for around 5 % of the traffic and others, like Microsoft and Akamai, among the ones with fastest growth.

Other works [58, 66] report a similar strong concentration (in 2012) when measuring from edge networks, with up to 65 % of the HTTP and HTTPS traffic going to the top 10 providers. We illustrate this trend in Figure 1.1. The remote IP addresses of all flows crossing the University of Twente (UT) border routers are translated into IP owners using the MaxMind GeoIP Organization [106] dataset. The top organizations exchanging traffic with the UT are then calculated. Two datasets are plotted: the first (Sept 2008, in Figure 1.1(a)) shows a flat distribution of traffic among several Internet Service Providers (ISPs); the second, captured four years later (Oct–Dec 2012, in Figure 1.1(b)), shows that the traffic at the UT has become much more concentrated around a few remote organizations, including the ones offering cloud services, such as Google, Akamai and Amazon.



(a) 2 weeks in Sept 2008

(b) Oct–Dec 2012

Figure 1.1: Top organizations exchanging traffic with the UT.

It is not surprising that many companies are considering to migrate services to the cloud [80]. Outsourcing to the cloud is deemed advantageous given the gains obtained from the reduced costs, flexible provisioning and high scalability. However, this migration also has several drawbacks. Cloud providers have been repeatedly related to reports of major failures [31]. Similarly, privacy of cloud services has been the center of an intense debate, owing to the possibility of direct access to users' private data by providers and, more alarmingly, foreign governments [7, 75].

In our view, potential dependability problems of cloud services will impel *enterprise customers* to look for assurances and validation of the performance promised by cloud providers. Hence, our first objective is to study simple and scalable methods for monitoring performance of cloud services, such that customers could monitor their services easily and independently. Privacy issues, on the other hand, will prompt the appearance of *private* and *national* cloud providers, and new players need knowledge about existing services to compete with international providers. Therefore, our second objective is to understand the implications of the design and usage of cloud services for the Internet, aiming to foster the development of new services.

This chapter is further organized as follows. Section 1.1 introduces the background on *cloud services* and motivates our scope. Section 1.2 details our goals, approach and research questions. Finally, Section 1.3 presents the thesis outline, whereas Section 1.4 lists the publications serving as basis for this thesis.

## 1.1 Cloud Services

This section introduces the fundamentals of cloud services. We start by presenting a definition for *cloud services* (Section 1.1.1), followed by their key characteristics (Section 1.1.2) and two examples (Section 1.1.3). After that, we provide examples of both Service Level Agreements (SLAs) offered by major providers and recent cases of dependability problems (Section 1.1.4) in order to illustrate that customers are in a weak position when migrating to the cloud. Finally, we analyze possible social and privacy issues related to the adoption of cloud services (Section 1.1.5).

### 1.1.1 Definition

*Cloud computing* and, by consequence, *cloud services* have been interpreted in several manners. For example, the services offered by cloud providers have been categorized according to what is delivered (*e.g.,* infrastructure, platform or software), the deployment model (*e.g.,* public, private or hybrid), among others. Multiple terms in the form *XaaS* – standing for *X as a Service* – can be found in the literature [123]. More often, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) are used to classify the different cloud offers [9, 54, 84, 144, 150], but some works go further, including even Human as a Service (HuaaS) [98] as a cloud service.

Simultaneously, cloud computing has become a hype. Thanks to this combination of a diversity of meanings with a business hype, a significant number of people does not recognize any novelties in the concept. For example, Richard Stallman has been quoted for his strong opinion against cloud computing [10]:

> *I think that marketers like cloud computing because it is devoid of substantive meaning [...] Perhaps the term "careless computing" would suit it better.*

We agree that the terms cloud computing and cloud services are overused as a business strategy to advertise technical solutions that have been mature already for a long time. Because of that, and to clearly delineate the scope of this thesis, we follow the conservative point of view of Armbrust et al. [9], and assume that cloud computing is simply the combination of software delivered as a service over the Internet (*i.e.,* SaaS) with *utility computing.* Utility computing is, in turn, the model of offering computing resources on demand, with customers being charged based on utilization [150]. Based on these concepts, a *cloud service* can be defined as follows:

**Definition 1** *A cloud service is any application that relies on utility computing to be delivered on demand over the Internet.*

This thesis focuses on studying the performance of cloud services from the customers' perspective, as illustrated in Figure 1.2. The figure depicts how Definition 1 is reflected into the relation between providers and customers. In this example, an IaaS or PaaS Provider offers utility computing (*i.e.,* either as infrastructure or as a development platform) to a SaaS Provider. Customers, which can be either enterprises outsourcing their applications or ordinary end users, pay the SaaS Provider in some form[1] for the services they use via the Internet. Note that we employ the term *cloud provider* throughout the thesis always referring to SaaS providers, except if the opposite is explicitly stated.

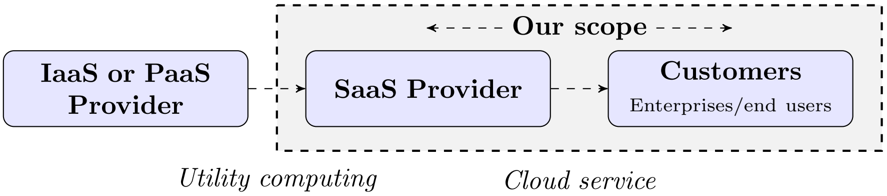


Figure 1.2: Providers and customers in a cloud environment (based on [9]).

### 1.1.2 Key Characteristics of Cloud Services

This section summarizes properties assigned to cloud services by recent surveys [9, 54, 144, 150], aiming to have a concise set of properties that characterize

[1] For example, customers may pay indirectly by receiving advertisements.

a cloud service. Naturally, we list only the ones meaningful given Definition 1. Five aspects can be considered key characteristics of cloud services:

- **Shared resources or multi-tenancy**: resources are shared among several customers in a cloud environment. In contrast, customers of conventional data centers normally do *not* share the same pool of resources.
- **Scalability, elasticity or dynamic provisioning**: users can allocate resources on-the-fly, without providers' assistance. For example, in a cloud storage service, customers can increase their storage space by requesting it from the pool of resources. Some authors refer to this property as "the appearance of infinity computing resources" [9].
- **Abstract infrastructure or virtualization**: cloud customers do not know the details of the infrastructure and systems providing the services, but instead, control them using well-defined interfaces. Note that abstraction and virtualization do not necessarily mean that virtual machines are in place – *e.g.,* as it is the case when a platform is offered as a service.
- **Pay-per-use or utility-based pricing**: although the units used to charge customers vary greatly, cloud services adopt the pricing model of a utility, with customers paying based on usage.
- **Connectivity, ubiquitous accesses or Internet centric**: by definition, cloud services are delivered via the Internet. As a consequence, private enterprise systems are not considered cloud services in this thesis.

Some works (*e.g.,* [144, 150]) list the existence of SLAs between customers and providers as a characteristic of cloud services. We do not agree with this. In fact, we align with Durkee [54] and Habib et al. [79] and argue that the lack of well-specified and comprehensive SLAs, which could be independently monitored and properly validated, limits the adoption of cloud services. Section 1.1.4 will provide examples of SLAs offered by popular providers.

### 1.1.3 Examples of Cloud Services

Cloud services are normally on-line alternatives to native applications. The most prominent case by the time of writing is *cloud storage* (*e.g.,* Dropbox [48] and Microsoft SkyDrive [110]), which can be considered a type of networked file system. In the particular case of Dropbox, for example, Amazon [4] provides utility computing, by means of the Amazon Elastic Compute Cloud (Amazon EC2) and the Amazon Simple Storage Service (Amazon S3), while Dropbox acts as the SaaS provider. It is easy to see that cloud storage is a typical cloud service satisfying all properties listed in Section 1.1.2. This thesis uses cloud storage as a case study, because, as it will be shown later, it is a popular application that already accounts for a major share of Internet traffic.

Similarly, cloud-based office suites seem to have all characteristics of a cloud service. Google Docs [71], which competes with the native Microsoft Office suite, is a well-known offer. However, since both computing power and the final service are controlled by the same organization, some of the properties listed in Section 1.1.2 are not immediately visible. It is hard to know, for example, how tenants share resources and how elasticity is supplied in Google Docs.

These examples show that, from the user's perspective, the differences between cloud services and conventional Web services are small. In fact, cloud services are a new way of offering services to end users via the Internet – *i.e.,* they are Web services offered as a utility. As more critical applications are provided in the cloud, however, issues of the cloud model become evident. The following sections discuss issues that motivate our research.

### 1.1.4 The Dependability of Cloud Services

Cloud providers have been involved in numerous performance incidents. A recent survey of media articles [31] reveals evidence of 49 outages in 20 providers worldwide (10 SaaS) during the 6 year period ending in 2011. The causes are various, ranging from power outages to software updates. Since the study has taken into account only events that received media attention, *the frequency of problems is likely to be much higher.* Moreover, such problems impact much more people than similar outages in private data centers, since many customers share resources in the cloud environment. New cases reported since then reinforce the findings. A variety of basic mistakes continue to appear among the causes, going as far as programming errors related to the leap year in 2012 [69].

Although it is often assumed that customers are backed by SLAs, current SLAs of cloud services are weak at best and, in general, written to protect the providers. Table 1.1 exemplifies the SLAs of some cloud offers. With this list, we do not aim at a comprehensive survey, but instead, we show how customers have very little protection when accepting the standard contracts of powerful providers. For illustration, we include examples of IaaS, PaaS and SaaS products, even though we focus only on the latter in the remainder of the thesis.

The table shows that some providers do not offer any guarantees. In turn, others include terms to make it harder for customers to request refunds. Amazon calculates violations on a monthly basis and only refunds a customer when (i) the customer has instances in more than one Availability Zone;[2] and (ii) *all*

[2] Availability Zones are independent and physically isolated parts of the Amazon Web Service (AWS) infrastructure.

Table 1.1: SLAs of some popular cloud offers.

| Provider | Promise | Violation Policy |
|---|---|---|
| Amazon EC2 | 99.95 % availability | The service is unavailable if *all* customer's instances have no connectivity in more than one Availability Zone. |
| Google Apps | 99.9 % uptime | Uptime is accounted in minutes per month. A service is down if it has 5 % of "user error rate" in 1 min. |
| Windows Azure | Per product | Each functionality has its own policy, with specific metrics. |
| Dropbox | Best effort | None |

customer's instances in at least two Availability Zones have no external connectivity. Google has a similar strict policy, accounting downtime only if the service has more than 5 % of "user error rate" (which is poorly defined in the contract) in a 1-minute interval. Furthermore, most contracts specify that customers must claim refunds. For example, in Microsoft's SLA it is written:

> *In order to be eligible to submit a Claim [...] the Customer must first have notified Customer Support of the Incident [...] within five business days following the Incident.*

While these terms might not be a problem to individuals who use the cloud for non-critical tasks, *enterprise customers* need assurances before migrating any essential application. Such contracts certainly do not offer enough guarantees. Moreover, customers do not always have technical means to validate the quality levels of a service without providers' interference. This motivates the first objective of this thesis (see Section 1.2).

### 1.1.5 Privacy and Social Implications

The biggest advantages of cloud services, such as the previously cited reduced costs and high scalability, are direct consequences of the multiplexing of customers' demands in a cloud environment. This concentration creates a centralized architecture, in contrast to the distributed origins of the Internet, which gives power to cloud providers and, eventually, results in gains of scale.

Paradoxically, the biggest issues surrounding cloud services are also outcomes of this concentration of power. Taking a social perspective, a cloud environment can be compared to the *Panopticon*,[3] used by Foucault [62] as a metaphor to describe how power and discipline are imposed in modern societies. The Internet has already been compared to a new *Panopticon* [6, 17], because ISPs, Service Providers and, ultimately, governments are able to observe and control people's activity without being noticed. Recent cases of privacy violations are clear examples of the Internet being used as a *Panopticon*, such as the alleged use of the Internet by the Chinese central government to control citizens and local governments [7] or the infamous data collection program of the United States National Security Agency (NSA) (known as PRISM [75]) that is alleged to receive information directly from American ISPs and cloud providers.

As in [57], we argue that cloud services push the Internet even further toward the *panopticism*, because companies and individuals are more and more entrusting their data to cloud providers, seduced by widely publicized advantages, but without any technical or legal means to safeguard their privacy and maintain a balance of power with providers. First signs that privacy violations can change these relations of power already start to appear, such as regional players entering the cloud market [95, 119] to offer both stronger privacy and protection against foreign governments, while worldwide firms hesitate to trust international providers [145]. This motivates our second objective (see Section 1.2).

## 1.2 Goals, Approach and Research Questions

### 1.2.1 Objectives

*Enterprise customers* outsourcing to the cloud are exposed to dependability problems. These customers will naturally look for guarantees before migrating any essential applications, which should include not only comprehensive contracts, but also methods for monitoring the services. Although some initial developments can be cited [32, 33, 79, 135], independent methods for customers to monitor performance of cloud services are still lacking. Therefore, the first objective of this thesis is:

**Objective 1**: *to investigate simple and scalable methods for monitoring performance of cloud services from the users' point of view, thus providing means for customers to monitor services in the cloud easily and independently.*

[3] The *Panopticon* [11] is a structure conceived to allow someone in a position of authority to observe, at any time, all inmates in a prison (or in any other hierarchical organization). The inmates, on the contrary, are not able to know whether they are being observed or not. This "state of conscious and permanent visibility" [62] ensures power and discipline automatically.

The high public interest in cloud services, together with the demand for alternative providers, already push new providers to enter the cloud market (*e.g.,* see [95, 119]). However, cloud services are relatively new, and very little is known about the workload they have to face, typical performance bottlenecks and, most of all, implications of different design choices. New players need such knowledge to compete with established providers in a timely manner. Therefore, our second objective is:

**Objective 2**: *to understand how cloud services are implemented and the implications of their design and usage for the Internet, thus providing guidelines for the development of new, well-performing cloud services.*

### 1.2.2 Approach

We follow a measurement-based approach founded primarily on the analysis of data *passively* collected from the *network*. As in any measurement-based study, (i) *what is measured*; and (ii) *how the measurements are taken* are the main ingredients of the approach. Both are described in the following.

#### What to Measure?

Each type of cloud service may be implemented and used differently and, therefore, may have its own peculiarities. Among the several cloud offers, we use *cloud storage* as a main case study. Cloud storage has been selected because it is becoming more and more popular, bringing cloud computing to people's daily routine and already generating a significant share of Internet traffic.

Furthermore, despite the many possible performance aspects that could be monitored, we study primarily *availability* and *responsiveness*, since those are, according to standards [146], the aspects perceived by end users. Availability is equally defined for any services, whereas responsiveness is application-specific [146]. Because of that, the thesis is divided into two parts: the first part concentrates on Objective 1 and studies a method for monitoring availability of *generic* cloud services. The second part, instead, extends our method to application-specific metrics (Objective 1) and provides an in-depth analysis of the design and implementation of cloud storage services (Objective 2).

#### How to Measure?

The use of *passive* measurements is a natural choice because our two objectives require information about real service usage. Alternative active methods, instead, rely on the injection of artificial requests [21]. Active experiments will, however, complement our analyses particularly when evaluating guidelines for the development of cloud storage services.

Since we search for simple and scalable passive monitoring methods, this thesis investigates to what extend cloud services can be monitored using *flow measurements* [19, 28, 138]. As Chapter 2 will discuss, devices for measuring flows are widely deployed and have been successfully employed in a variety of applications that require scalable and privacy-preserving ways for collecting data in high-speed networks [131].

Two other options for collecting passive measurements have been considered and discarded. Firstly, *server instrumentation* is out of scope since providers and customers are assumed to have conflicting interests in our scenario and, thus, providers cannot be trusted to be the only source of measurements. Indeed, when cloud services suffer from performance degradations, cloud monitoring applications often become unavailable as well [31]. Secondly, *client instrumentation* has been discarded because we are looking for methods that are scalable and easy to deploy. The complexity of installing client-side monitoring agents is increasing as "the era of personal computers installed with a large number of different applications is coming to an end" [129]. Therefore, instrumenting client devices tends to become harder, or at least less convenient, than measuring at fewer network vantage points.

### 1.2.3 Research Questions

Our two objectives together with the chosen approach lead to the research questions addressed in this thesis.

Firstly, a strong motivation for using flow measurements to monitor cloud services is the pervasiveness of devices with flow export capabilities – *i.e.,* flow-based methods could be immediately deployed, relying on equipment already in place for other applications. Our first research question, therefore, aims to investigate whether popular measurement devices are suitable for our goals:

1. *Are popular flow-based measurement devices suitable for serving as data source for monitoring cloud services?*

Secondly, although the idea of employing flow measurements to monitor cloud services is intuitively appealing when compared to alternatives such as client instrumentation, taking a flow-based approach implies the use of approximations, since flow measurements are known to be unrelated to high-level metrics usually employed to report the performance of applications [151]. With the next research question, we aim at both developing a systematic method to monitor performance of cloud services and evaluating the suitability of such a flow-based approach:

2. *Are flow measurements suitable to monitor cloud services? What are the limiting factors for such an approach?*

The remaining research questions are directly related to Objective 2 and the selection of *cloud storage* as our case study. Firstly, we apply the method developed while answering the previous question to understand typical usage and performance bottlenecks of Dropbox – the most popular cloud storage provider by the time of writing:

3. *What are the typical usage and performance characteristics and bottlenecks of Dropbox?*

Finally, we complement our study of cloud storage services with a series of *active* experiments, in which we compare how different providers implement cloud storage, highlighting implications of design choices:

4. *How do different providers implement cloud storage services and what are the implications of the design choices for client performance?*

## 1.3 Thesis Organization

This thesis is organized in two parts. We start from a broad scope, evaluating the use of flow measurements to monitor *generic* cloud services, and move to an in-depth analysis of *cloud storage* services. The chapters in each part are depicted in Figure 1.3 and summarized in the following.

### Part I – Generic Cloud Services

Part I will focus on Objective 1 only, and evaluate the use of popular flow export technologies to monitor generic cloud services. This part is divided into two chapters as follows.

**Chapter 2 – Understanding Flow Data Sources** – will study whether popular flow measurement devices are suitable to monitor performance of cloud services, thus answering Research Question 1. Literature study is combined with active experiments to determine the consequences of different implementations and measurement artifacts on flow datasets. Results in Chapter 2 are a reference on how flow measurement devices should be evaluated prior to their usage in any flow-based application.

**Chapter 3 – Monitoring Cloud Services using NetFlow** – will introduce a simple method to monitor availability of cloud services using NetFlow, the most popular technology for measuring flows by the time of writing. The method

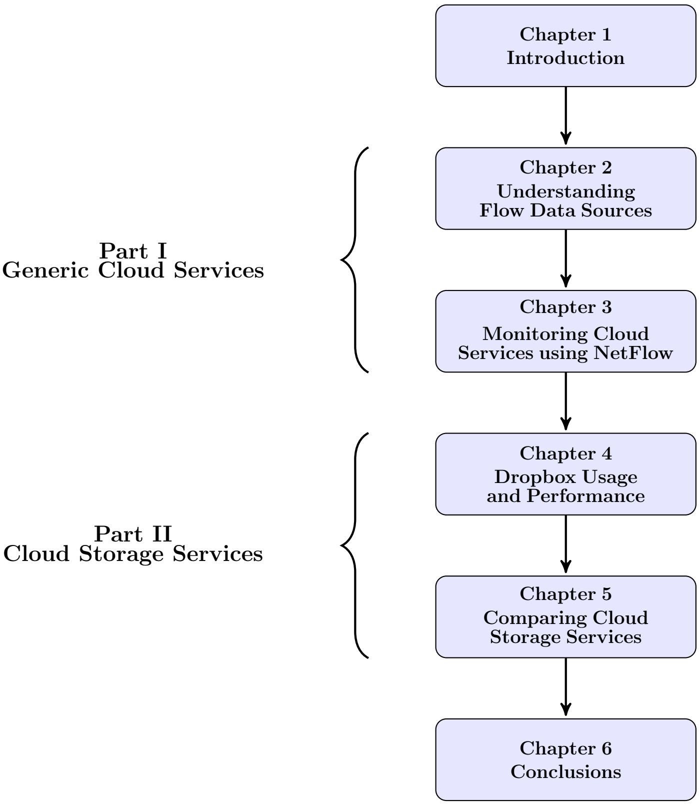


Figure 1.3: Thesis organization.

is prepared to cope with both sampled and non-sampled NetFlow data and, therefore, it targets high-speed networks. Two case studies are then used to evaluate the flow-based approach, partly answering Research Question 2.

## Part II – Cloud Storage Services

Part II will present an in-depth analysis of cloud storage services. This part is also composed of two chapters as follows.

**Chapter 4 – Dropbox Usage and Performance** – will provide the results of the first comprehensive characterization of usage and performance of Dropbox. First, the chapter extends our method for monitoring cloud services to application-specific metrics, complementing our answer for Research Question 2. Then, flow data collected in different countries by mean of specialized devices are used to evaluate typical workloads and performance bottlenecks of Dropbox (Research Question 3).

**Chapter 5 – Comparing Cloud Storage Services** – will analyze how cloud storage services are implemented and study the impact of different designs on performance (Research Question 4). This is achieved by (i) introducing a methodology to study both system architecture and client capabilities of cloud storage services; and (ii) executing a series of benchmarks. Chapter 5 contributes with guidelines on how well-performing cloud storage services should be implemented.

Finally, **Chapter 6 – Conclusions** – concludes the thesis, summarizes our contributions and lists future works.

## 1.4 List of publications

The complete list of papers published during the four years of my Ph.D. can be found in the appendices (see "About the Author"). Among those, the following publications have been used as basis for this thesis:

- Drago, I. and Pras, A. 2010. *Scalable Service Performance Monitoring.* In Proceedings of the 4th International Conference on Autonomous Infrastructure, Management and Security, AIMS'10. 175–178. **Chapter 1**.

- Hofstede, R., Drago, I., Sperotto, A., Sadre, R., and Pras, A. 2013. *Measurement Artifacts in NetFlow Data.* In Proceedings of the 14th International Conference on Passive and Active Measurement, PAM'13. 1–10. **Best Paper Award of PAM 2013**. **Chapter 2**.

- Drago, I., Hofstede, R., Sadre, R., Sperotto, A., and Pras, A. 2013. *Measuring Cloud Service Health using NetFlow/IPFIX: the WikiLeaks Case.* Journal of Network and Systems Management. Accepted for publication. **Chapter 3**.

- Drago, I., Mellia, M., Munafò, M. M., Sperotto, A., Sadre, R., and Pras, A. 2012. *Inside Dropbox: Understanding Personal Cloud Storage Services.* In Proceedings of the 12th ACM Internet Measurement Conference, IMC'12. 481–494. **Awarded with an IETF/IRTF Applied Networking Research Prize 2013**. **Chapter 4**.

- Drago, I., Bocchi, E., Mellia, M., Slatman, H., and Pras, A. 2013. *Benchmarking Personal Cloud Storage.* In Proceedings of the 13th ACM Internet Measurement Conference, IMC'13. **Chapter 5**.

# Part I

# Generic Cloud Services

CHAPTER 2

# Understanding Flow Data Sources

Flow export technologies, like Cisco NetFlow [27] and the standardization effort IPFIX [126], are already widely deployed. They owe this success to their widespread integration into network devices. The pervasiveness of these technologies has resulted in their use in a variety of application areas that go far beyond simple network monitoring, such as flow-based intrusion detection [131] and traffic engineering [35]. Using existing flow data sources to monitor cloud services is a natural next step that could immediately assist organizations that want to monitor services that are outsourced to the cloud.

Flow export is a complex process that includes the real-time aggregation of information about packets into flows and the periodic export of *flow records*. Although there exist standards defined by the Internet Engineering Task Force (IETF) to export flow information from network devices (*i.e.,* IPFIX), the IETF has intentionally avoided the specification of *flow exporters*, in order to broaden the applicability of the standardized protocols [138]. Moreover, flow exporters are known to be affected by measurement artifacts [139], which can reduce the quality of flow data substantially. Both the peculiarities of different flow exporters and possible measurement artifacts need to be taken into account when performing any flow-based analysis.

The main goal of this chapter is to study whether popular flow measurement devices are reliable for serving as data source for monitoring cloud services. We achieve this goal in two steps. Firstly, we document *how changes on parameter settings of flow exporters impact flow data.* For answering this question, we revisit the IPFIX Request for Comments (RFCs), summarizing the standard recommendations for measuring and exporting flows. We then complement the literature survey with active experiments, testing several measurement setups and highlighting effects on the obtained datasets. Secondly, we analyze *to what extend measurement errors of popular devices would harm the monitoring of cloud services.* We answer this second question by means of a case study. Active experiments and flow data analysis are combined to assess whether our own flow exporters would provide data of sufficient quality to our monitoring goals.

The knowledge gained by answering these questions helps to understand whether differences in flow data are caused by (i) normal variations of measurement settings; or (ii) measurement errors. While the former needs to be compensated for before using the data, the latter may impair the analysis permanently. The lessons learned from this chapter are valuable as guidelines on the design of flow-based applications and assist in determining whether or not a particular data source is appropriate for the specific monitoring task.

This chapter is organized as follows. Section 2.1 discusses related work. Section 2.2 provides background on flow monitoring. Section 2.3 describes the measurement methodology used to answer both questions. Section 2.4 studies how different settings of flow export parameters affect flow data. Section 2.5 evaluates the quality of our flow exporters in a case study, assessing their suitability for monitoring cloud services. Finally, Section 2.6 concludes the chapter.

## 2.1 Related Work

This chapter provides both the background on flow-based monitoring and a case study on how to assess the suitability of flow exporters for a particular application. Flows have already been used in a variety of applications: network security, intrusion detection and application identification are some examples intensively researched in recent years [39, 44, 102]. Each of these applications has its own requirements and may react differently when exposed to measurement artifacts. Our methodology to evaluate the quality of flow exporters is generic and, therefore, can be applied to any other flow-based applications as well.

Some works present a general discussion about the importance of calibrating measurement devices [21, 90, 117]. The recommended calibration steps include checking for clock inaccuracies and for data loss during the several monitoring stages. Focusing on the use of flows for monitoring cloud services, this chapter performs such steps, highlighting the implications of both common measurement errors and normal variations of parameter settings.

Other works focus on measurement artifacts found in particular situations. In [34], Juniper exporters are shown to suffer from problems when routing updates are performed. Time intervals in which no flows are measured can be observed in flow time series during these events. Artifacts resulting from the interception of packets via port mirroring are described in [149]. Similarly, the limitations of using commodity hardware for capturing packets are analyzed in [65]. Even though some of these artifacts may occur in specific devices only, researchers and operators need to be aware of them to anticipate impacts and build robust analysis applications. Our work goes in a similar direction, shedding light on new artifacts found in widely deployed flow exporters.

## 2.2 Background on Flow Monitoring

Flow-based monitoring originates from the need for measurement systems able to provide a detailed view on network traffic [19, 138]. In comparison, flow measurements have a finer granularity than what is normally obtained with the Simple Network Management Protocol (SNMP), but a higher aggregation than full packet captures. Because of this aggregation, flow monitoring requires less processing and storage than full packet captures. Hence, flow monitoring is more scalable and can be performed at higher network speeds.

The deployment of flow monitoring technologies started in the 1990's with both the Real-time Traffic Flow Measurement (RTFM) protocol [18, 20] (which is based on SNMP) and the introduction of various proprietary systems, like Cisco NetFlow [27]. As noted in [19], however, the ideas of flow monitoring also had been presented in academic works (*e.g.,* [26]). In the early 2000's, the IPFIX Working Group was formed in the IETF, to standardize protocols to export IP flows. While originally targeting some key applications [122, 151], such as accounting and traffic profiling, flow data have proven to be useful for several network management activities, going as far as being used for Voice over IP (VoIP) [5, 37] and DNS [38] traffic monitoring, among others.

We summarize the essential background on flow monitoring in the following. Using the IPFIX RFCs as a reference, we first introduce the definition of a *flow* and the basic terminology in Section 2.2.1. The IPFIX architecture for monitoring flows is summarized in Section 2.2.2, in which we also position our contributions. After that, Section 2.2.3 discusses general IPFIX guidelines on measuring flows. Finally, Section 2.2.4 compares IPFIX and widely used versions of Cisco NetFlow.

### 2.2.1 The Definition of a Flow

This thesis assumes the definition of a flow as established by the IETF in [28]:

> *A Flow is defined as a set of IP packets passing an Observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties.*

Some key concepts complement the definition. *Observation points* are the places in the network where packets are intercepted, such as network interfaces of a router, optical splitters or shared Ethernet media. Observed packets are grouped into flows by means of a (variable) set of properties, called the *flow key.* Flow keys can include (i) any fields in the packet headers up to the application layer – *e.g.,* IP addresses and port numbers; (ii) characteristics of the packets

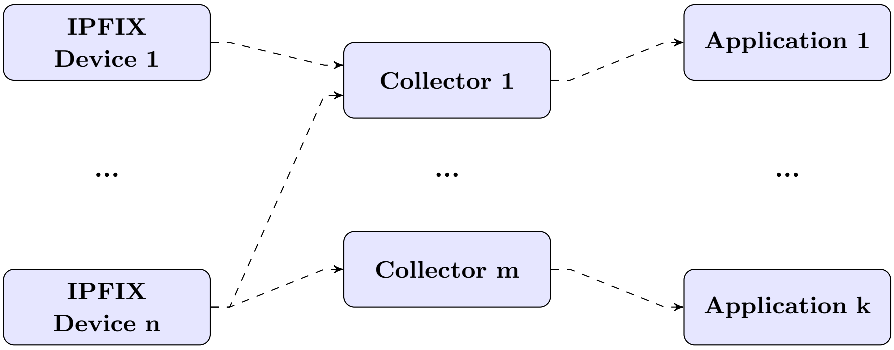


Figure 2.1: IPFIX reference architecture (source [126]).

– *e.g.,* payload sizes; or (iii) the outcome of processing the packets – *e.g.,* next hop IP addresses and application identifiers [29].

A *flow record* carries the observed properties of a flow. These properties are called *information elements* and encompass both key and *non-key* fields. As for the flow key, non-key fields can also include information from several protocol layers (*e.g.,* TCP flags), packet characteristics (*e.g.,* number of MPLS labels) etc. The basic list of information elements is maintained by the Internet Assigned Numbers Authority (IANA) in the IANA IPFIX Information Element Registry [88]. Enterprise-specific information elements can be defined too, allowing new fields to be specified without any alterations to the protocol or to the IANA's registry. Flow record formats (*i.e.,* the list of information elements) are exchanged by different components of the IPFIX architecture (described next) through standardized *templates.*

### 2.2.2 Flow Monitoring Architecture

A reference architecture for measuring flows is presented in [126] and depicted in Figure 2.1. Three main components are part of this architecture:

- *IPFIX devices* measure flows and export the corresponding records. IPFIX devices include at least one *exporting process* and, normally, observation points and *metering processes.*

  Metering processes receive packets from observation points and maintain flow statistics. Exporting processes encapsulate flow records and control information (*e.g.,* templates) in IPFIX messages, and send the IPFIX messages to flow collectors.

- *Collectors* receive flow records and control information from IPFIX devices and take actions to store or further process the flows.

- *Applications* consume and analyze flows – *e.g.,* the previously mentioned intrusion detection systems are typical flow-based applications.

Figure 2.1 shows that IPFIX devices, collectors and applications exchange data with multiple peers. The decoupled architecture of IPFIX provides scalability gains for flow-based applications, since data are processed and aggregated at each stage. Typically, each component in Figure 2.1 is hosted independently, with IPFIX devices being installed directly inside routers, switches, or dedicated probes placed nearby network edge nodes. IPFIX devices are also called *flow exporters* – *e.g.,* when only exporting processes are present. We employ the terms *flow exporter* and *IPFIX device* indistinguishably throughout this thesis.

The main contributions of this thesis are in the use of flows for monitoring cloud services. Our work assumes that flow records are the primary source of information and focuses on the last part of the IPFIX architecture, *i.e.,* on new analysis applications. Since any flow-based application requires flow data of good quality to perform its tasks satisfactorily, it is essential to understand how flows are measured in practice. The next section describes general IPFIX recommendations related to flow exporters.

### 2.2.3 IPFIX Devices

Figure 2.2 illustrates the typical tasks of an IPFIX device [126]. Firstly, packets are captured in an observation point and timestamped by the metering process. Then, the metering process can apply functions to sample or filter packets. Sampling and filtering techniques are specified in the context of the Packet SAMPling (PSAMP) protocol [152]. Filters select packets deterministically based on a function applied to packet contents. Samplers, in contrast, combine such functions with techniques to randomly select packets.

Sampling and filtering play a fundamental role in flow monitoring because of the continuous increase in network speeds. By reducing the amount of packets to be examined by metering processes, sampling and filtering make flow monitoring feasible under higher network speeds. On the other hand, these techniques might imply loss of information, which restrict the usage of flow data substantially. The effects of sampling and filtering will be summarized in Section 2.4.2.

Packets that pass the sampling and filtering stages update entries in the flow cache, according to predefined templates. Flow records are held in the cache until they are considered expired, following the reasons described next. Expired records are made available by the metering process to exporting processes, where they are combined into IPFIX messages and sent out to flow collectors.

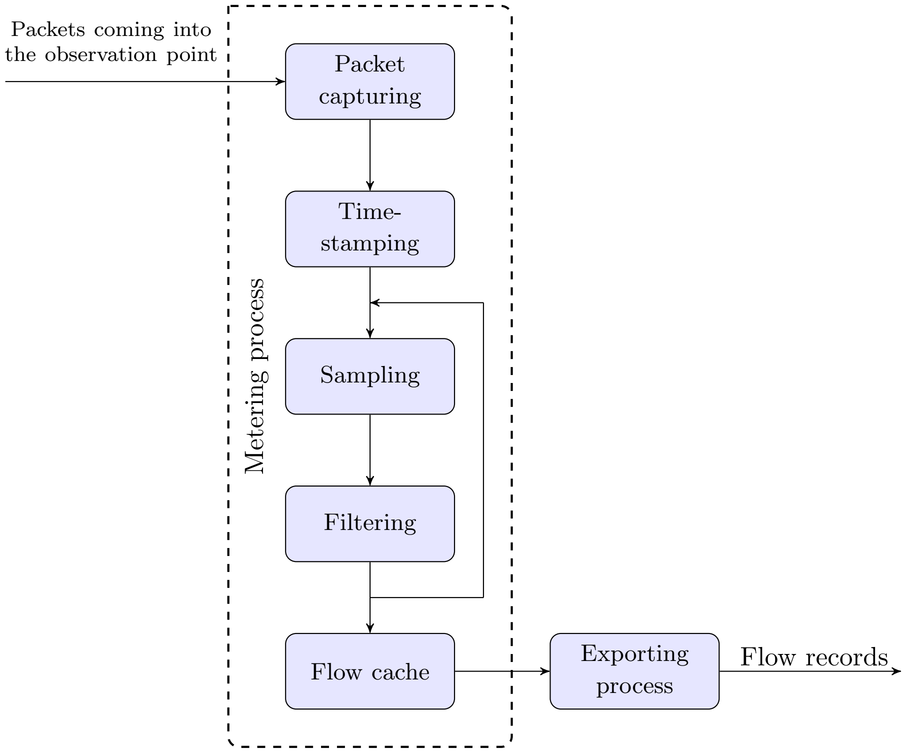


Figure 2.2: Functions performed by an IPFIX device (based on [126]).

### Flow Expiration

Flow records are usually expired from the flow cache by metering processes based on given timeout parameters or when particular events are detected. IPFIX standards, however, do not mandate precise cases in which records need to be expired and exported. IPFIX does provide, instead, guidelines on how metering processes should expire flow records [126]:

1. *Idle timeout*: No packet belonging to a flow has been observed for a specified period of time;

2. *Active timeout*: The flow has been active for a specified period of time. Therefore, the active timeout helps to report the activity of long-lived flows periodically;

3. *Lack of resources*: Special heuristics can be used to expire flow records prematurely in case of resource constraints in the IPFIX device. For example, IPFIX devices can change timeout parameters at run-time when facing high network loads, to prevent the flow cache from being exhausted.

Other reasons to expire records can be found in practical implementations of flow exporters. Cisco NetFlow exporters often rely on heuristics to determine the end of a flow – *e.g.,* packets with `FIN` or `RST` flag set terminate TCP flows before the idle or the active timeouts are triggered [27]. Some Cisco NetFlow exporters also rely on special timeout parameters, called *fast aging*, to expire short flows faster. The impact of expiration policies on flow data will be evaluated in Section 2.4.1.

### 2.2.4 Relation to NetFlow

IPFIX and popular versions of NetFlow differ in the used flow export formats. NetFlow version 5 (v5) provides fixed flows – *i.e.,* flow fields cannot be changed. The fixed format considerably limits the applicability of NetFlow v5, since no protocol evolution is possible. NetFlow v5, for example, cannot be used to monitor IPv6 traffic. However, several references (e.g., [39, 115]) suggest that NetFlow v5 is still the most widely deployed flow export protocol by the time of writing and, therefore, it is an important source of flow information.

NetFlow v9 overcomes the limitations of NetFlow v5 by allowing flexible flow configuration via templates. IPFIX design has started from NetFlow v9 [97]. Both IPFIX and NetFlow v9 support the export of generic fields (*i.e.,* information elements). IPFIX, however, adds new functionalities, such as structured data formats for information elements [30], the reliable transport of flow records between exporters and collectors etc. Major differences on protocol messages are also found when comparing IPFIX to NetFlow v9. Although relevant for the development of flow exporters and collectors, these differences will not be discussed further, since they are not important in our context.

We emphasize that this comparison refers only to export protocols, and not to particular products. When considering how flows are *defined* by particular *exporters*, other differences can be cited. Cisco NetFlow v5 exporters, for example, rely on a constant flow key, composed of 7 attributes: source and destination IP addresses and port numbers, IP protocol number, IP type of service and input interface index. Newer Cisco exporters support the export of configurable NetFlow v9 records, but without allowing flow keys to be freely defined. Such capabilities have been included in recent Cisco exporters under the label *IOS Flexible NetFlow*, in contrast to what Cisco nowadays calls *Original NetFlow*. *IOS Flexible NetFlow* exports flexible flows using either NetFlow v9 or IPFIX. Interested readers can find a comparison of Cisco exporters in [24].

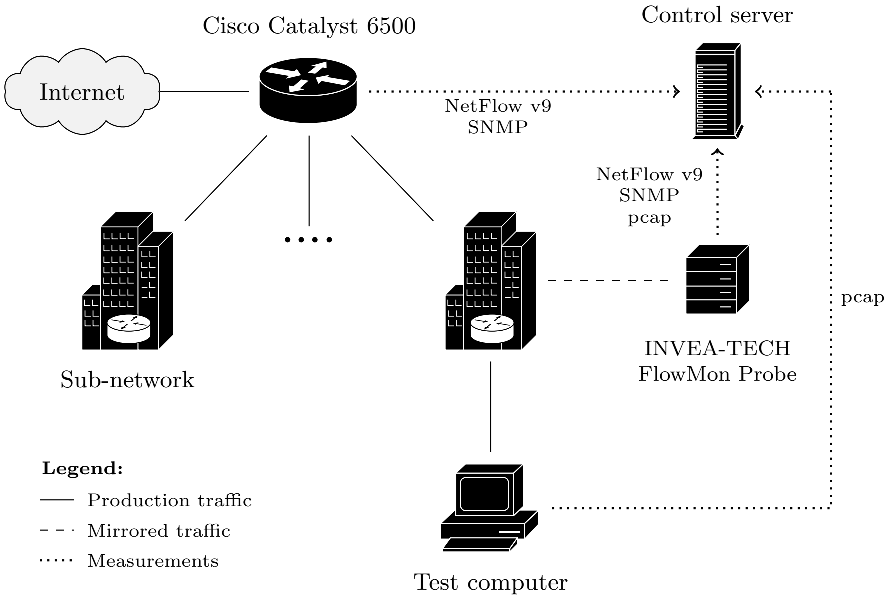


Figure 2.3: Infrastructure to study flow data quality.

## 2.3 Measurement Methodology

Figure 2.3 depicts the setup we use to study flow data quality as well as to illustrate the behavior of flow exporters under different parameter settings – note the line marks in the figure. The measurement environment is operational in our network. It is composed of 2 flow exporters (*i.e.,* a device from the Cisco Catalyst 6500 series and a dedicated INVEA-TECH FlowMon Probe), 1 control server that collects measurements from other devices and 1 test computer.

The monitored network is connected to the Internet via a Cisco Catalyst 6500.[1] This device is particularly representative, because the Cisco Catalyst 6500 is one of the most widely deployed switching platforms [61], found in many service provider, enterprise and campus networks. Several departmental networks are directly connected to the Cisco Catalyst 6500. The traffic of one of these sub-networks is also mirrored to an INVEA-TECH FlowMon Probe. Finally, a test computer is used to inject traffic in the network in active experiments, which are observed at both flow exporters.

[1] Our specific device is configured with the `WS-SUP720-3B (PFC3B, MSFC3)` hardware modules and the `IOS 12.2(33)SXI5` operating system.

Measurements are collected from both flow exporters and from the test computer by a single control server. The two flow exporters send NetFlow v9 records to a flow collector (NFDUMP [78]) installed in the control server. Management information is collected from both flow exporters as well, by means of SNMP agents. Finally, packet headers (*i.e., pcap* files) can be captured in both the INVEA-TECH FlowMon Probe and in the test computer.

The environment is used in two sets of experiments. Firstly, Section 2.4 illustrates the effects of parameter settings of flow exporters, aiming to document the *normal variations* that analysis applications should expect on flow datasets. Secondly, Section 2.5 evaluates the quality of flow data exported under realistic traffic conditions, unveiling *measurement errors* found in the devices deployed in our environment. More details about the experiments are provided next, together with the respective results.

## 2.4 The Impact of Parameter Settings

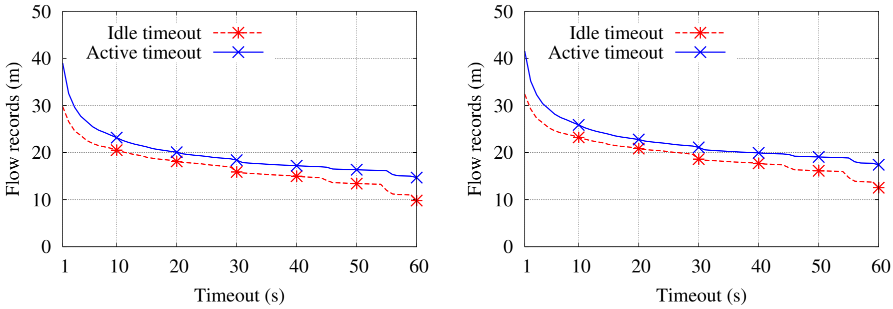
The IPFIX recommendations (see Section 2.2.3) suggest that typical flow exporters will allow at least the following to be configured:

- Flow templates, defining the list of exported flow properties;
- Flow expiration policies, including active and idle timeouts and, depending on the implementation, special heuristics to expire flows faster;
- Sampling and filtering functions, including filtering rules, sampling algorithms, and related parameters (*e.g.,* the sampling probability).

Indeed, both exporters in our measurement environment provide such options, even though they are not compliant to IPFIX. Since NetFlow v5 is still the most widely used protocol to export flows [39, 115] and we aim at reusing flow data in Part I of the thesis, we only consider the basic fields also found in NetFlow v5.

The impact of other parameters is illustrated by means of off-line experiments, in order to test several flow export settings. We capture a dataset of packet headers in our dedicated FlowMon Probe during 24 hours (see Figure 2.3).[2] We then use YAF [89] to convert the packet headers to flow records under different setups. YAF is a flow exporter that can be easily installed and customized. We have extended YAF to allow us to control how flow records are expired and whether sampling is used. Moreover, IPFIX-specific functionalities in YAF, such as bidirectional flow export [137], have been disabled. The effects of expiration policies as well as sampling and filtering mechanisms are discussed in Section 2.4.1 and Section 2.4.2, respectively.

[2] Only TCP traffic is analyzed, since most cloud services are built on top of TCP [64, 94].



(a) Without checking TCP flags

(b) Checking TCP flags

Figure 2.4: Impact of expiration policies on the number of exported flow records.

### 2.4.1 Expiration Policies

Expiration policies control the period of time that flow records are kept in the flow cache. Because flow exporters might implement different policies, or even change settings at run-time to cope with high network loads, it is important to understand the consequences of varying expiration policies.

Figure 2.4 depicts the total number of exported flow records after YAF processes our dataset. Different heuristics to expire flows are tested. In Figure 2.4(a), flow records are expired only by means of timeout parameters – *i.e.,* either idle or active timeout. In Figure 2.4(b), instead, TCP flags are also used to expire flows, as in Cisco exporters – *i.e.,* besides the expiration by timeouts, a packet with `FIN` or `RST` flag set causes the flow record to be expired. The effects of varying idle and active timeouts are shown in separate lines in both figures. For each case, we vary the respective timeout while the expiration by the other timeout parameter is disabled.

By comparing Figure 2.4(a) to Figure 2.4(b), we can see that using flags to expire flow records results in slightly more records. Moreover, both figures allow us to conclude that the number of flow records varies considerably when either the idle or the active timeout is changed. Using larger timeout values results in a higher aggregation of packets into flow records – *i.e.,* less flow records are exported when timeouts are increased. For example, the number of exported records decreases around 35 % when the idle timeout is increased from 10 s to 50 s in Figure 2.4(a). Note also the sharp increase in the number of records when timeout parameters are lower than 10 s.

Although exporting less flow records is generally positive, to reduce the usage of both the network and flow collectors, larger timeout values increase flow cache utilization in the exporter. Figure 2.5 illustrates the maximum number

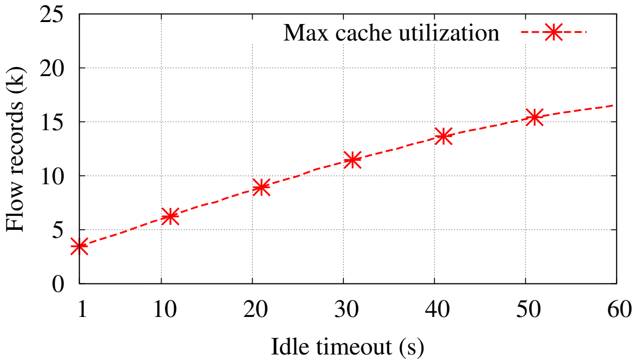


Figure 2.5: Maximum flow cache utilization for different idle timeouts.

of flow records in YAF cache when our dataset is processed under different idle timeouts – all other expiration policies are disabled in this example. We see that the maximum number of active flows increases with the idle timeout. YAF reports a maximum of 5,990 active flows in its cache when the idle timeout is 10 s, but the number more than doubles (15,257) when the idle timeout is changed to 50 s. Section 2.5 will show that high flow cache utilization may lead to measurement errors. Moreover, as the utilization approaches the cache capacity during peaks of traffic in the network, the flow exporter might change expiration policies automatically to prevent the cache from being exhausted. Such automatic changes, however, are usually *not* informed to flow collectors and to analysis applications.

These results show that the same traffic stream can result in different flow records, depending on expiration policies. Chapter 3 will study methods to monitor availability of cloud services. Intuitively, one could assume that a sharp change in the number of flow records could be an indication of either service problems or measurement errors. Figures 2.4 and 2.5 demonstrate that such changes might simply be a consequence of flow exporters adapting expiration policies at run-time, as a reaction to high cache utilization. Flow-based applications, therefore, cannot trust *raw flow records* blindly. They need to normalize the data to compensate for the effects of expiration policies, in order to be robust against different export settings.

### 2.4.2 Sampling and Filtering

The effects of sampling and filtering are well-documented [152]. Because filters are deterministic, their consequences can easily be understood: only a well-defined subset of the original packets are measured. If filters are applied to flow keys, the relation is even more explicit, with only a part of the original *flows* being measured. Filtering will be used in several experiments in this thesis to isolate a subset of packets for specific analyses – *e.g.,* as in Figure 2.4, in which we isolate only the TCP traffic in the dataset.

Sampling, on the other hand, selects a random subset of packets for flow accounting, thus impacting not only the number of flow records, but also all flow properties (*e.g.,* observed packets, bytes and TCP flags). For the sake of brevity, we do not present the results of combining expiration policies with sampling, since similar conclusions to those in the previous section would be obtained – *i.e.,* as in the non-sampled case, *raw packet-sampled flow records* are not appropriate for monitoring cloud services.

Methods to estimate the original number of flows, packets and bytes from packet-sampled flow records have been described extensively in [49, 50, 51, 52]. If packets are sampled independently with probability $p = 1/N$, some properties of the original data stream (*e.g.,* the number of packets) can be estimated by rescaling the measured quantities by a factor $N$. More elaborate estimators, which make use of the observed TCP flags, are required for estimating the original number of flows. Chapter 3 will rely on the previous work to post-process flow datasets and compensate for the effects of sampling while monitoring cloud services. However, these methods are effective only if flow exporters are not affected by measurement errors, as we will discuss next.

## 2.5 Measurement Errors

We now discuss the experience acquired while calibrating our flow exporters. This thesis assumes only two basic requirements for considering a flow data source appropriate for monitoring cloud services:

1. The flow exporter reports all flows in the network or, if sampling and filtering are applied, it adheres to setup parameters. As such, we assume that possible variations in flow datasets are solely an outcome of parameter settings of flow exporters;

2. We assume that flow properties represent the information observed in the original packets correctly. For example, flow records usually report information about the TCP flags and the number of packets seen in the network. We assume that packet counters are precise, and all flags of the original packets are taken into account and reported in flow records.

Based on these two assumptions, the remaining chapters will focus on methods to map the low level flow measurements into performance metrics that are meaningful at higher protocol layers.

The goal of the experiments in this section is to verify whether the measurement devices in our network satisfy these requirements. Our exporters are tested for the first requirement in Section 2.5.1. The second requirement is analyzed in two parts. Section 2.5.2 checks whether exported time information is accurate. Section 2.5.3 discusses problems found in the remaining flow fields.

Both device documentation and personal communication with operators and vendors have been used to understand the causes of the identified problems. Note that the list of measurement errors presented in this section is by no means comprehensive, since errors are *load- and configuration-dependent*. The results in the following, instead, illustrate the importance of checking prerequisites before employing flows in any flow-based application.

### 2.5.1 Missing Flows

Our first experiment checks whether all flows in the network are reported by the flow exporters. All results in this section have been obtained by collecting SNMP measurements and flow records while the devices were handling the traffic of our production network (see Figure 2.3). We use both proprietary Management Information Bases (MIBs) to monitor the status of flow caches and standard MIBs to monitor the number of packets in the network. The SNMP measurements are then compared to the information in flow records.

Our measurements reveal that both exporters miss flows. However, while the dedicated INVEA-TECH FlowMon Probe misses flows rarely, because of well-known problems such as packet loss in the monitored link, the Cisco Catalyst 6500 presents a serious measurement artifact, related to how the device handles high cache utilization. Both problems are described and compared in the following.

#### Cache Utilization and Flow Learn Failures

Our Cisco Catalyst 6500 fails to monitor all flows when its flow cache utilization is high. In such situations, several time intervals in which no flows are measured (gaps) can be observed. This happens because active flows are stored in a cache of limited size in the Cisco Catalyst 6500 (128 k entries in our equipment). The position of flows in the cache is determined by hashing flow keys. This Catalyst model supports a maximum of two hash collisions, and colliding hashes are stored in a second cache of 128 entries – *i.e.,* only two flows with different keys leading to the same hash value can be accommodated simultaneously, up to a

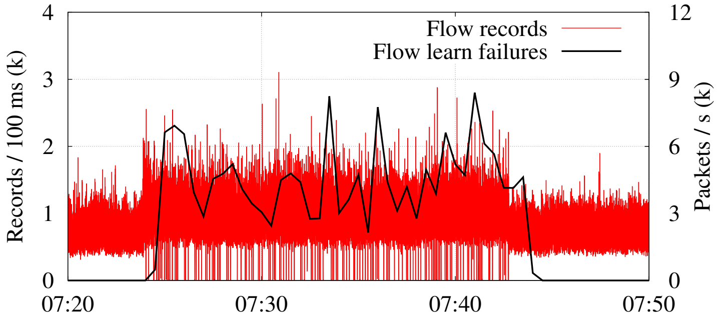


Figure 2.6: Impact of flow learn failures on flow time series.

maximum of 128 collisions. When a packet belonging to a new flow cannot be accommodated, a *flow learn failure* happens. The total number of flow learn failures can be monitored using Cisco's proprietary MIBs.

We use the Cisco's MIBs to monitor flow learn failures and describe how the artifact is manifested in flow data. Such results can help to understand whether the artifact is present in a dataset, without having access to administrative interfaces or SNMP agents of exporters. Our experiments show that the first packets of flows are more likely to be subject to flow learn failures, since subsequent packets of flows already in the cache are matched until a record is expired. Therefore, smaller flows are more frequently missed, while larger flows might have only their first packets missed. Moreover, initial control packets (*e.g.,* TCP `SYN` packets) more likely evade the monitoring. As we will show in the coming chapters, such packets are very important for our monitoring goals.

Figure 2.6 shows a time series of the number of flow records exported by our Cisco Catalyst 6500 in intervals of 100 ms. These data have been collected early in the morning, when the device normally starts to run out of flow cache capacity because of the increase in traffic during business hours in our network. A constant stream of flow records without gaps can be observed until around 7:25 AM, when the number of records increases (see the left-hand $y$-axis). Simultaneously, flow learn failures (right-hand $y$-axis, in packets/s) start to be reported by the SNMP agents, and short gaps appear in the time series of flow records – *i.e.,* the time series of flow records reaches zero in several short time intervals. Note that the two time series in the figure are slightly out of phase because the SNMP measurements are reported in a 5 min granularity only.

Interestingly, the gaps caused by flow learn failures are periodic, especially when the network load causes the flow cache utilization to be constantly close

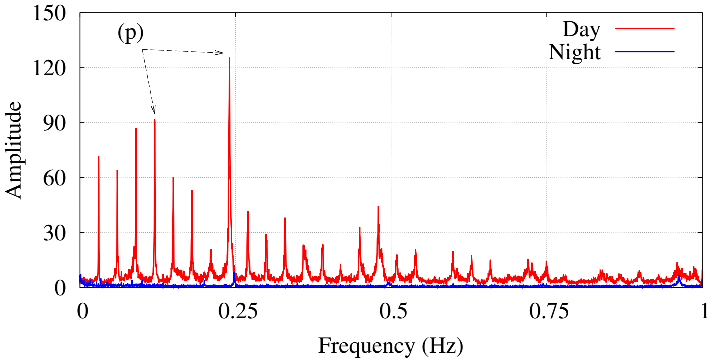


Figure 2.7: Fourier transform of the time series of flow records exported by our Cisco Catalyst 6500.

to the cache capacity. When analyzing data of this device for 2 weeks, we observe that the distribution of the time between gaps is strongly concentrated around multiples of 4 s. Moreover, the gaps are not bigger than 2 s in 95 % of the cases. The periodicity can be further confirmed by applying the Fourier transform to the time series of the number of flow records exported per time interval – 500 ms bins are used for illustration. Figure 2.7 shows the frequency components for both diurnal and nocturnal traffic. The traffic of each day in our 2-week dataset is processed separately, and the obtained spectra are averaged to improve visualization [16]. Spikes at the frequency corresponding to 4 s (*i.e.,* 0.25 Hz) and at sub-harmonics (*e.g.,* 0.125 Hz) can be observed in the diurnal traffic, when flow learn failures are very common – see the mark (p) in Figure 2.7. The same behavior is, on the other hand, not seen in nocturnal traffic. This periodic pattern suggests that the gaps are an outcome of a cyclic process, responsible for expiring flow records from the cache.

The behavior of an exporter under high flow cache utilization is naturally dependent on the way the exporter is implemented. When the same analysis is performed with our INVEA-TECH FlowMon Probe, other artifacts emerge. This device also locates flows in the cache by hashing flow keys, but hash collisions are handled in software using linked lists. As such, the device is not subject to flow learn failures. Under high load, the device releases cache space by exporting flow records earlier, *i.e.,* by ignoring timeout parameters. Since all packets are still reported, this artifact can be compensated for by analysis applications, as we will show in Chapter 3. Under very extreme conditions, however, the device may suffer from the effects of packet loss, which are described next.

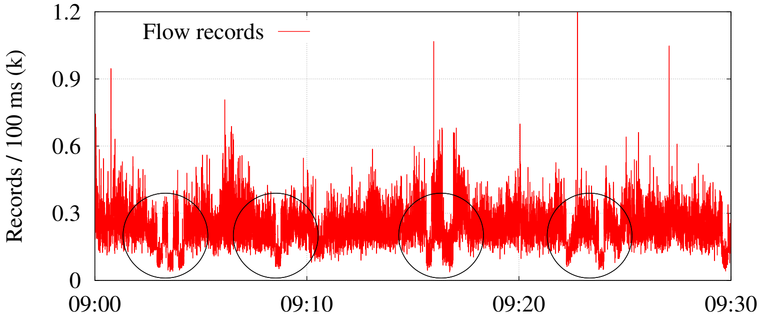


Figure 2.8: Impact of NetFlow packet loss on flow time series.

### Packet Loss

Flows can also be missed because of packet loss between exporters and collectors, packet loss in the monitored link etc. In contrast to flow learn failures, these problems tend to occur randomly, resulting either in a homogenous reduction in the number of flow records or in short non-periodic gaps.

Figure 2.8 illustrates this effect by plotting a time series of flow records exported by the INVEA-TECH FlowMon Probe, during a period in which the collector has been intentionally overloaded with data-intensive tasks. The information collected via SNMP agents confirms that more than 5 % of the NetFlow packets have been lost by the collector during this interval. Several short periods with a reduced number of flow records can be seen (marked with circles), but without any strong patterns in this case.

The loss of NetFlow or IPFIX packets can be identified at the collector side directly, using the sequence numbers present in both NetFlow and IPFIX packet headers. As such, collectors and analysis applications can check whether the data are complete. Packet loss in the monitored link, on the other hand, is more harmful: it is equivalent to applying sampling without an explicit choice of the sampling strategy [50, 51]. If the loss rate is high while using flows to monitor performance, problems can stay undetected, leading to erroneous conclusions.

## 2.5.2 Timing Errors

Previous studies [2, 60] exemplify that the response time of popular Web services ranges from several milliseconds to a few seconds. This raises the question of whether flow data provide time information with sufficient accuracy for monitor-

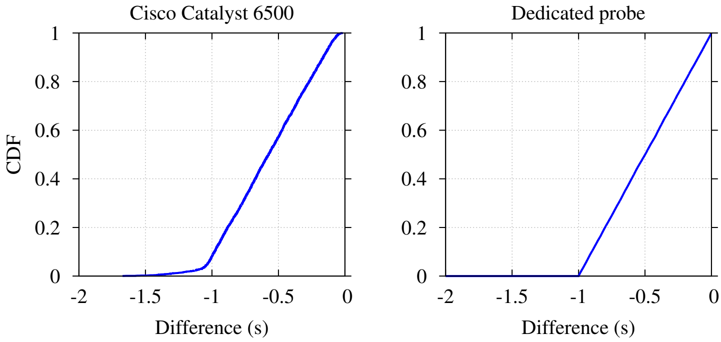
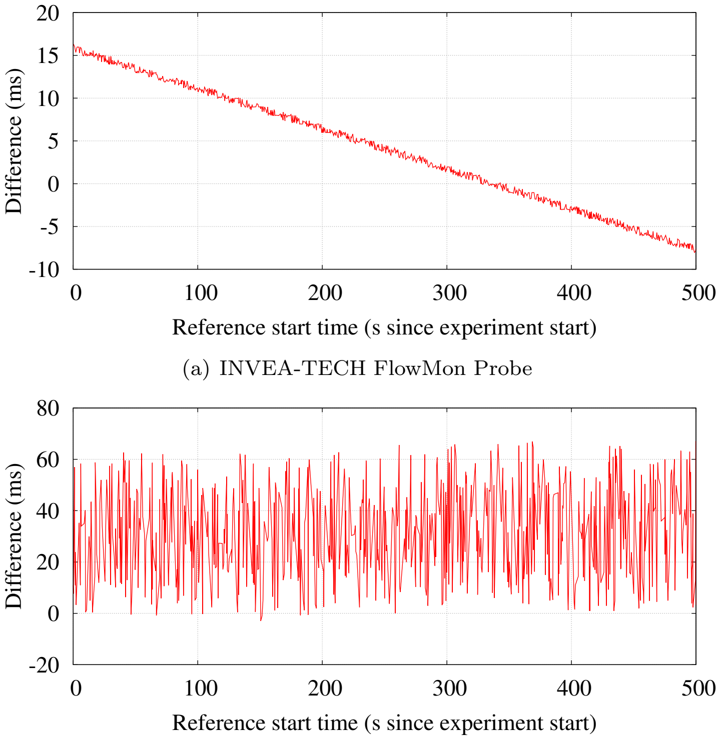


Figure 2.9: CDF of the difference between flow start times measured at the test computer and at the flow exporters.

ing typical services. NetFlow exports timestamps with millisecond resolution. IPFIX exporters, in turn, can use special information elements to report up to nanoseconds. However, the fact that a time attribute is exported in a specific *precision* does not guarantee that the information has such an *accuracy* [117].

The existence of timing errors in flow measurements has already been discussed in [92, 139]. Two types of errors were described: (i) a cyclic error originating in the design of NetFlow v9; and (ii) errors that are dependent of the flow exporter implementation, such as clock skew and export delays. The combination of these errors results in a surprisingly poor accuracy. Indeed, time fields exported with NetFlow v9 have an accuracy of *seconds* only. Although this error can be fixed at the collector side, popular collectors, such as NFDUMP [78], normally process and archive time fields as they are exported, turning the millisecond information unusable.

An example of the poor accuracy of NetFlow v9 is provided in Figure 2.9. We use the test computer (see Figure 2.3) to inject traffic and create several flows in the network, which are observed at both flow exporters as well. Flow start times are then recorded at the different devices. Figure 2.9 reports the CDF of the difference between the start time of each flow measured at the test computer and at the flow exporters. Since the Round Trip Time (RTT) between the devices is very low, one expects only a small time difference between the measurements of a single flow, because of clock synchronization, for example. However, Figure 2.9 shows a high error in both devices, almost uniformly distributed on the interval [-1, 0] s, as expected given the conclusions in [92, 139]. These results make clear that, independently of the flow exporter, raw timestamps exported by NetFlow v9 do not have accuracy appropriate for our monitoring goals.



(a) INVEA-TECH FlowMon Probe

(b) Cisco Catalyst 6500

Figure 2.10: Corrected start times compared to values measured at the test computer. Note the $y$-axes.

Other NetFlow versions as well as IPFIX do not suffer from this cyclic error. However, implementation-specific problems cannot be overlooked. Figure 2.10 presents results of repeating the active measurements with our test computer after implementing and deploying a method for correcting the NetFlow v9 cyclic error in our collector (see [139]). The figure depicts time series of the differences in flow start times observed during the first 500 s of the experiment.

We can see in Figure 2.10(a) that the dedicated probe exports stable time information, which slowly drifts away from the reference clock at the test computer. This behavior can be ignored in our case because the error caused by the clock drift is negligible – *i.e.*, only few milliseconds per minute would be added to reported time fields. Figure 2.10(b), on the other hand, shows a random error of around 60 ms on the Cisco Catalyst 6500, confirming the results of [139]. Such error is indeed higher than some quantities we will evaluate in the remaining chapters (*e.g.,* the response times of some cloud storage services) and, thus, it turns this Cisco Catalyst 6500 inappropriate for our analyses.

### 2.5.3 Imprecise Flow Fields

#### Flag Information

Flow records normally include a summary of the flags observed in TCP flows. The analysis of records exported during the active experiments discussed in the previous section reveals a simple, but serious, measurement error in our Cisco Catalyst 6500.

The Cisco Catalyst 6500 implements two mechanisms to decide how packets are forwarded in the network [25]. Most packets are processed by specialized hardware to improve the system performance when taking forwarding decisions (*i.e.,* packets are hardware-switched). When forwarding decisions cannot be taken in hardware, the packets are processed in software by the generic CPU of the device (*i.e.,* packets are software-switched). Software decisions are considered exceptions and happen, for example, when packets with expired time-to-live counters or IP packets with header options are processed.

The NetFlow exporter in our Catalyst 6500, however, does not measure flag attributes when packets are hardware-switched. Software-switched packets are, instead, rightly measured. Since most packets are hardware-switched, this artifact implies that only few TCP flows are exported with flag information. Even more, in contrast to what is specified in the documentation of this device [25], TCP flags *do trigger* the expiration of flow records, precisely as discussed in Section 2.2.3. As such, TCP flags are observed and considered in the flow expiration process, even though they are not reported in all flow records.

The lack of flag information is problematic for several applications. For example, many works rely on flags for inferring statistics from packet-sampled flows [50, 51, 76]. Chapter 3 will present our method to handle packet-sampled flows, which also relies on TCP flags to track the status of cloud services. None of these applications can be deployed when TCP flags are not properly measured.

#### Byte Counters

By injecting packets of several sizes using our test computer and evaluating the byte counters of obtained flow records, we observe that the Cisco Catalyst 6500 also exports wrong byte counters for hardware-switched flows. This problem happens because the NetFlow exporter does not strip the padding bytes of small IP packets that are transported as Ethernet payload. The problem affects all frames that carry less than 46 bytes.[3]

The impact of this artifact depends on the fraction of Ethernet frames that carry less than 46 bytes. Using the packet header traces captured at the dedicated FlowMon Probe (see Section 2.4), we verify the potential damage caused by this problem. Our results show that, although 20 % of the frames carry less than 46 bytes, the exporter would report only around 0.2 % more bytes than the correct numbers, in total. Therefore, the practical consequences of the artifact can be neglected in most cases.

### Other Artifacts

Other artifacts observed in our devices are described in [85] and include: (i) the imprecise implementation of flow expiration by active and idle timeouts; and (ii) inconsistencies in non-TCP flow records, which are sometimes exported with TCP flags set. The first artifact is harmless for our work, since its effect is identical to varying timeout parameters of flow exporters at run-time (see Section 2.4). The second artifact may lead to misconceptions and errors – *e.g.,* if an application uses only flag information to filter specific TCP flows, non-TCP flows can be wrongly filtered. Since the remaining chapters will only evaluate TCP flows, this artifact can also be safely ignored in our context.

## 2.6 Conclusions

The first part of this thesis aims at using flow data to monitor generic cloud services. A major motivation for that is the widespread deployment of network devices with flow export capabilities: flow measurements are, therefore, readily available. However, it is crucial to understand whether such data have sufficient quality for the intended analysis. In this landscape, the contributions of this chapter are twofold.

Firstly, we revisited the basic background on flow monitoring and highlighted the implications of varying parameter settings of flow exporters. Using packet traces collected in our network, we showed that flow datasets can differ considerably, even when no measurement errors are present. For instance, IPFIX standards suggest exporters to change timeout parameters at run-time when facing high loads. Such changes may result in an unpredictable number of flow records. Thus, analysis applications need to be prepared to compensate for that, in order to handle data exported under different conditions.

[3] Ethernet payloads should have a minimum size of 46 bytes. If an IP packet is smaller than 46 bytes, padding bytes are added to the frame (see [85, 92]).

Secondly, using our own flow exporters in a case study, we identified and analyzed measurement errors occurring in flow data. The errors are related to missing flows, wrongly measured properties and inaccurate time information.

While our dedicated exporter (an INVEA-TECH FlowMon Probe) was shown to report flows of sufficiently good quality, the second analyzed device (a specific device from the Cisco Catalyst 6500 series) had proven inappropriate for our analysis. Our Cisco Catalyst 6500 presents a residual timing error higher than what we expect for the response time of some typical services. Moreover, it does not export all required information for handling packet-sampled flows, and misses most flows when the network traffic load exceeds a certain limit.

Such results show that, although flow data are indeed readily available, not all flow data sources are appropriate for advanced applications. Operators and researchers, therefore, must carefully check and calibrate their data sources before deploying any new flow-based application.

CHAPTER 3

# Monitoring Cloud Services using NetFlow

The advantages of cloud services, such as reduced costs, easy provisioning and high scalability [9, 54], attract more and more enterprise customers. We see an increasing interest from organizations in migrating services like file storage or e-mail to cloud providers [80]. However, such migration also has its drawbacks. Cloud services have been repeatedly related to major failures [31], including data center outages, loss of connectivity and performance degradation. Enterprise customers have, therefore, a vital need to monitor cloud services that are essential for their businesses, both to identify and report problems to providers and to validate whether promised quality levels are indeed satisfied.

The goal of this chapter is to investigate to what extend the performance of cloud services can be monitored using only NetFlow. As discussed in Chapter 2, flow measurements have been successfully employed in a variety of applications that require scalable ways to collect data in high-speed networks, such as to detect network intruders [131], to perform traffic engineering [35] and to discover connectivity problems in production networks [68, 128]. NetFlow, in particular, is a widely deployed technology for measuring flows [39, 115], often available at enterprise and campus networks [132]. Therefore, NetFlow seems to be an alternative to provide *immediate* and *scalable* means for customers to monitor their services in the cloud, without the burdens of instrumenting end user devices and, more importantly, without any interference from cloud providers.

However, although the use of NetFlow to monitor cloud services is intuitively appealing, NetFlow measurements are known to be unrelated to high-level metrics usually employed to report the performance of applications [151], such as *availability* and *response time*. Moreover, our results in Chapter 2 showed that flow datasets can be affected seriously by changes of parameter settings of flow exporters. For example, enabling packet sampling or tuning timeout parameters is often necessary, in order to cope with the traffic loads of high-speed networks, and such changes considerably affect the exported flow data. Both systematic methods for calculating performance metrics from NetFlow records and – more importantly – an assessment of such a flow-based approach are still lacking.

This chapter studies the use of NetFlow to monitor cloud services by proposing a method that (i) normalizes NetFlow data exported under different settings; and (ii) calculates a simple performance metric to indicate the *availability* of cloud services. We then evaluate the method by means of two case studies. First, non-sampled flow data collected in our network during 10 consecutive weeks are used to analyze performance anomalies in popular services. Second, packet-sampled flows collected at an international backbone are used to verify whether targeted services have been affected by the cyber-demonstrations organized during the *WikiLeaks Cablegate* [105].

Note that we restrict our scope to availability problems in this chapter because our goal in Part I of the thesis is to look for solutions that are generic, instead of application-specific. For the same reason, we refrain from considering any application-specific knowledge while developing our method in this chapter. Other performance aspects will be considered in Part II, when studying cloud storage services.

This chapter is further organized as follows. Section 3.1 describes our proposed method. Section 3.2 presents our first case study, in which performance anomalies in popular services are evaluated using non-sampled flow data. Section 3.3 presents our second case study, in which packet-sampled flows related to the *WikiLeaks Cablegate* are analyzed. Section 3.4 discusses the lessons learned from the two case studies, which will serve as guidelines for extending our method in Part II. Related work is described in Section 3.5 and Section 3.6 concludes the chapter.

## 3.1 Method

Assuming that a set of routers is exporting flow records while handling the traffic from a group of end users, we aim to use these data to monitor cloud services. A deployment scenario is depicted in Figure 3.1. Such setup is already very common, thanks to the popularity of network devices with flow export capabilities. In this example, users in an enterprise network interact with two cloud services (*i.e.,* Service A and B in the figure). The link between the enterprise network and the Internet is monitored by a flow exporter. The exporter processes the traffic mirrored from the network and forwards flow records to collectors, from where the records are accessible to analysis applications.[1] This example makes clear that a single enterprise does not have access to all traffic related to the services, but it can still observe a significant fraction, generated by its own users.

[1] See Chapter 2 for the background on flow monitoring, including details of the flow monitoring architecture standardized in IPFIX.

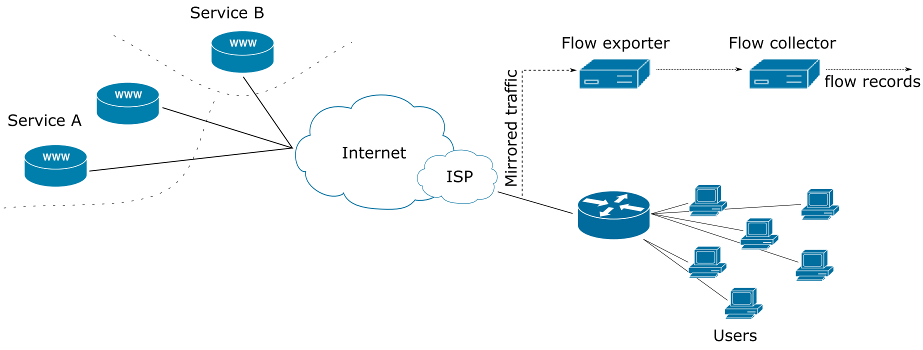


Figure 3.1: Envisaged deployment scenario for our method.

Similarly to [68, 128], we assume that users trying to contact unavailable services will generate traffic with particular characteristics that can be exploited to reveal the status of the cloud services. We call such traffic *unhealthy*. Then, we propose a method to estimate a *health index*, which we define as *the fraction of healthy traffic to a service in a given time interval*. Our method is summarized in Figure 3.2. It receives batches of NetFlow records from flow collectors and outputs the *health index* for selected services by performing the following steps:

- The traffic related to the services is filtered using either predefined lists of server IP addresses or names of IP owners, as determined by the MaxMind GeoIP Organization [106].

- The data are normalized to remove the effects of parameter settings of flow exporters – *e.g.,* the effects of packet sampling and flow expiration policies (see Section 2.4). This is achieved by estimating the total number of TCP connections $n$ and the number of *healthy* TCP connections $n_h$ for each service. We perform the mapping of flow records up to the transport layer, instead of the application layer, because this chapter aims to be generic.[2] We consider a TCP connection *healthy if it could exchange transport layer payload* – *i.e.,* if the connection has a complete TCP handshake.

  Given the major differences between non-sampled and packet-sampled flow records, different steps are followed by our method in each case. These steps will be described in Section 3.1.1 and Section 3.1.2, respectively.

- The *health index* is determined as the fraction of *healthy* TCP connections in the network – *i.e.,* $\frac{n_h}{n}$ is returned for each service.

[2] Other transport layer protocols are not evaluated because previous work [64, 94] already showed that TCP is the predominant transport protocol used in cloud services.

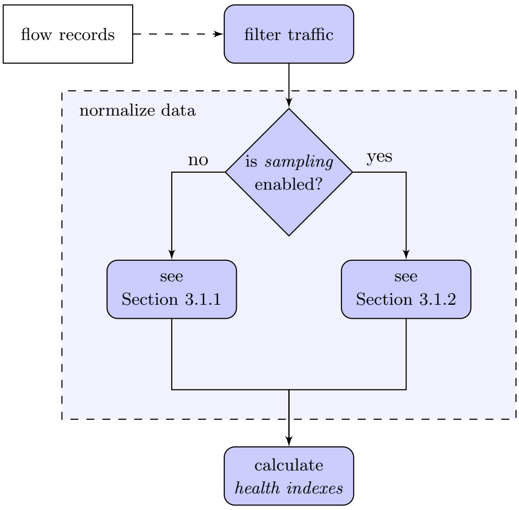


Figure 3.2: Measuring the *health index* from NetFlow data.

The *health index* is correlated to the service availability as follows. When a service is fully available, client TCP connections are expected to be normally established, making the index to approach 1; when the service becomes partially or fully unavailable, failed client connection attempts decrease the index. Note that this definition provides an upper bound for determining the actual service status. The *health index* decreases only if a service is unavailable. A high index, however, does not guarantee that a service is operating, since the method does not take application layer errors into account.

Next, we describe how the method handles non-sampled and packet-sampled flows in Section 3.1.1 and Section 3.1.2, respectively. After that, Section 3.1.3 describes the prototype that will be used in the case studies later in this chapter.

### 3.1.1 Non-Sampled Flow Data

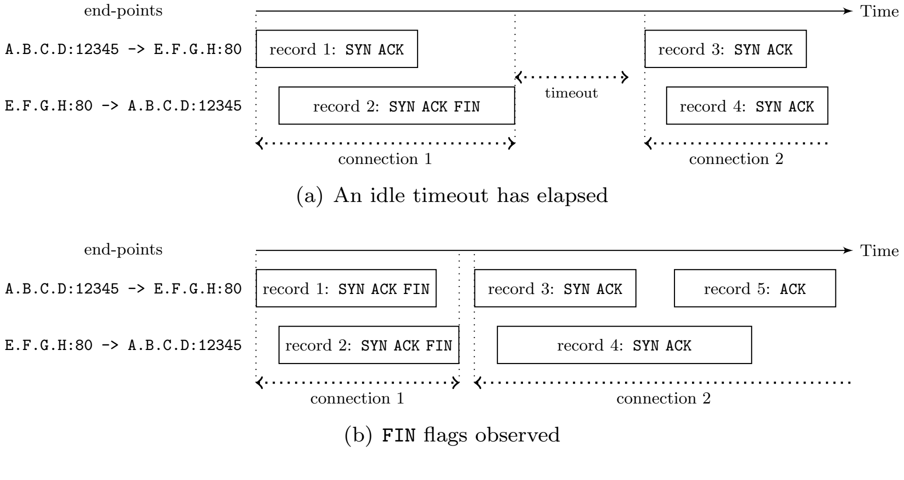
When packet sampling is disabled, all packets are taken into account, and flow records provide a summary of the observed TCP flags. Since different TCP connections, normally, are *not* reported in a single flow record, it is possible to monitor the status of TCP connections by identifying the flow records that belong to each connection [130]. Hence, our method executes two steps to normalize non-sampled flow data: (i) records that report information about the same TCP connection are aggregated; (ii) the information about TCP flags are evaluated to determine $n$ and $n_h$.

### Aggregating Flow Records

We rely on an extension of the heuristic presented in [130]: flow records with identical keys – *i.e.,* sharing the same IP addresses and port numbers – are merged until the original TCP connection is considered complete. Since NetFlow is usually unidirectional, flow records from both traffic directions are also aggregated. The critical step for the heuristic is to determine when a TCP connection is complete, such that later records sharing the same key start a new connection. In [130], a TCP connection is complete if: (i) an idle timeout has elapsed; and (ii) records with TCP `FIN` flag set have been observed from both end-points and a short timeout has elapsed.

Figure 3.3 illustrates how the heuristic aggregates flow records. In Figure 3.3(a), two TCP connections between the same end-points generate four flow records – note the TCP flags. After sorting the records by start time, the heuristic aggregates the first two records. Because the remaining records start after an idle timeout has elapsed, they become part of a new connection. Figure 3.3(b) illustrates the second rule: in this case, records with `FIN` flag set are seen from both end-points, thus signaling the connection termination. Therefore, the new records sharing the same key are part of a new TCP connection, even if the time interval between the TCP connections is shorter than the previous timeout parameter.

This heuristic is shown to produce good results in general, but the authors of [130] also show that it does not count TCP connections precisely when appli-



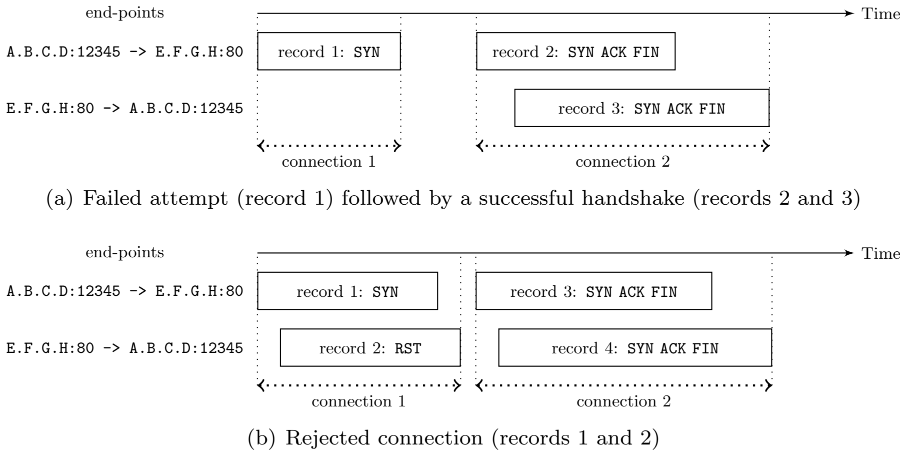
(a) An idle timeout has elapsed

(b) `FIN` flags observed

Figure 3.3: Aggregating flow records (based on [130]).

cations reuse sockets immediately. For example, in Figure 3.3(a), the two TCP connections would be wrongly merged if the interval between them is smaller then the selected timeout.

These problems, however, seem possible to be avoided by taking other TCP flags into account when aggregating the records. Indeed, inspired by the TCP analyzer implemented in Bro [116],[3] we improve the heuristic by extending the cases in which TCP connections are marked as complete. For example, we close an active connection if flow records indicate that the end-points have performed a new TCP handshake. Furthermore, we consider a TCP RST packet in either traffic direction sufficient for reducing the timeout that stops the aggregation of records. These two cases are depicted in Figure 3.4(a) and Figure 3.4(b), respectively.



(a) Failed attempt (record 1) followed by a successful handshake (records 2 and 3)

(b) Rejected connection (records 1 and 2)

Figure 3.4: Extensions to the heuristic to aggregate flow records.

Note that, analogous to [130], timeouts are still involved in the aggregation of flow records. We again rely on the reasoning of Bro [116], in which different timeouts are in place for (i) *idle* connections – *e.g., connection 1* in Figure 3.3(a); (ii) *complete* connections – *e.g., connection 1* in Figure 3.3(b) and Figure 3.4(b); and (iii) *attempts* without a reply – *e.g., connection 1* in Figure 3.4(a).

[3] Bro is a stateful network monitor that contains a module for analyzing TCP connections. By considering complete TCP headers, Bro is able to follow the TCP state-machine and determine final connection states precisely.

### Evaluating the Aggregated Records

The total number of TCP connections $n$ as well as the number of *healthy* connections $n_h$ can be trivially counted from the aggregated flow records. More precisely, all TCP connections are counted as *healthy*, except for:

- Failed attempts – *i.e.,* when clients send a TCP `SYN` packet and do not obtain any reply – see *connection 1* in Figure 3.4(a);

- Rejected TCP connections – *i.e.,* when servers reset connections without completing the handshake – see *connection 1* in Figure 3.4(b).

We refer to Appendix A for results showing that the steps to normalize non-sampled flow records presented in this section are consistent in several flow export scenarios – *i.e.,* the aggregated records match with the original TCP connections even if settings of flow exporters are changed.

## 3.1.2 Packet-Sampled Flow Data

Our method normalizes flow data exported under sampling as follows. Firstly, it estimates the total number of TCP connections $\hat{n}$ and the number of *healthy* TCP connections $\hat{n}_h$ directly from the raw packet-sampled flow records. Owing to sampling, those numbers are realizations of random variables. In order to compare the quantities in a meaningful way, confidence intervals are determined and taken into account: if the intervals overlap, or if $\hat{n}_h > \hat{n}$ by chance, there is not enough evidence of *unhealthy* TCP connections in the network – therefore, we make $\hat{n}_h \leftarrow \hat{n}$ and the index is reported to be 1; otherwise, the *health index* is calculated using $\hat{n}$ and $\hat{n}_h$.

### Estimating the Number of Connections

Methods to estimate the original number of TCP connections from raw packet-sampled flows are described in [52]. If packets are sampled independently with probability $p = 1/N$, $\hat{f} = Nk_S$, where $k_S$ is the number of observed records with `SYN` flag set, is proven to be an unbiased estimator for the number $f$ of TCP flows, under the assumption of one single `SYN` packet per TCP connection.

We rely on a similar reasoning to estimate the number of *healthy* TCP connections. Because *healthy* connections must have a complete TCP handshake, we assume that:

- *Healthy* TCP connections have exactly one `SYN` packet from both originators and responders;

- *Unhealthy* TCP connections have exactly one SYN packet from originators, but none from responders.

Let $k_{orig}$ and $k_{resp}$ be the number of observed flow records with SYN flag set from originators and responders, respectively, in a time interval. Then, $\hat{n} = Nk_{orig}$ and $\hat{n}_h = Nk_{resp}$ are unbiased estimations of the total number of TCP connections $n$ and of the number of *healthy* TCP connections $n_h$ in that time interval.

### Calculating the Confidence Intervals

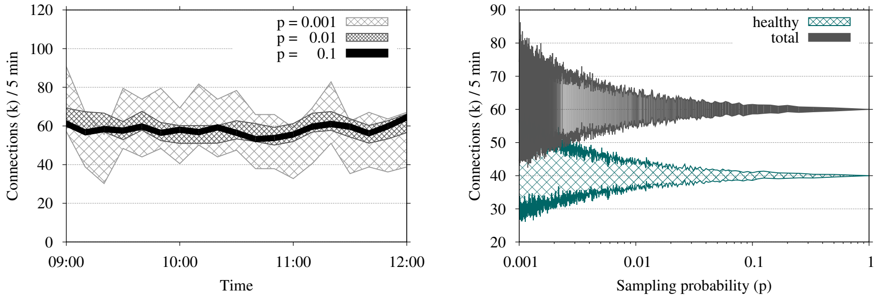
The expected values of $\hat{n}$ and $\hat{n}_h$ are equal if a service is available. However, a system can only be considered *unhealthy* if the difference between $\hat{n}$ and $\hat{n}_h$ is statistically significant. We evaluate the difference between $\hat{n}$ and $\hat{n}_h$ by estimating the probability distributions of $n$ and $n_h$. If confidence intervals for a given level of significance show a high probability that $n_h < n$, then $\hat{n}$ and $\hat{n}_h$ are returned. Otherwise, we make $\hat{n}_h \leftarrow \hat{n}$ and the service is considered *healthy.*

The basic idea is that we can only observe a flow record with SYN flag set if a SYN packet has been sampled. Sampling SYN packets forms a finite sequence of Bernoulli trials with success probability $p = 1/N$. If $n$ TCP connections occur within a fixed time interval, the number $k_{orig}$ of sampled records with SYN flag set from originators follows the Binomial distribution $f(k|n,p) = \binom{n}{k}p^k(1-p)^{n-k}$. If $n_h \leq n$ TCP connections are *healthy* in the interval, the number $k_{resp}$ of sampled records with SYN flag set from responders follows the Binomial distribution $B(k|n_h,p)$.

Given a series of observations $\mathbf{k_{orig}} = (k_{orig,1}, k_{orig,2}, ..., k_{orig,r})$ and $\mathbf{k_{resp}} = (k_{resp,1}, k_{resp,2}, ..., k_{resp,r})$ over $r$ time intervals, the parameters $n$ and $n_h$ of the Binomial distributions can be estimated using the method presented in [45]. The method assumes that $n$ and $n_h$ are constant over $r$ consecutive time intervals. In our context, this is a reasonable approximation for highly loaded services in short time intervals. Given the success probability $p$ and the observations $\mathbf{k_{orig}}$, the posterior distribution of $n$ is determined by:

$$f(n|\mathbf{k_{orig}},p) \propto (1-p)^{rn} f_0(n) \prod_{i=1}^{r} \frac{n!}{(n-k_{orig,i})!}, \tag{3.1}$$

where $f_0(n)$ is a prior distribution of $n$. If no prior knowledge about $n$ is available, the uniform distribution is used. The mode of $f(n|\mathbf{k_{orig}},p)$ is a biased estimation for $n$, with maximum likelihood. The confidence interval for $n$ can be calculated numerically [134], for instance, by computing $f(n|\mathbf{k_{orig}},p)$ for $n = max(k_{orig,i}), \ldots, M$ such that, for a given value $\delta$, $f(n > M|\mathbf{k_{orig}},p) < \delta$. After that, the probabilities of neighbor values of the mode are summed up until the total probability is higher than the specified significance level. The same



(a) Intervals for $n$ when applying different $p$

(b) Intervals for a fixed pair $n$ and $n_h$

Figure 3.5: The effects of varying the sampling rate.

steps are taken to calculate the posterior distribution of $n_h$ from the observations $\mathbf{k_{resp}}$, and, based on that, the confidence interval for $n_h$.

Since Equation (3.1) requires the calculation of large factorials, approximations may be necessary. Depending on the values of $n$ and $p$, the Normal or Poisson approximations for the Binomial distribution can be used. A practical usage example of the Normal approximation can be found in [133]. As for the previous section, we refer to Appendix A for results validating that the procedure in this section provides reliable estimations of $n$ and $n_h$.

### Example

We provide an example of how the method operates with packet-sampled flows. The sampling probability $p$, set in the flow exporter, is the most critical variable in this case, because confidence intervals calculated by Equation (3.1) depend on $p$. Sampling less packets implies wider intervals, which makes it harder to conclude that the difference between $\hat{n}$ and $\hat{n}_h$ is significant.

Figure 3.5(a) shows the effect of the changing the sampling probability $p$. We use part of the packet header dataset captured for the experiments in Chapter 2 (see Section 2.4) to illustrate intervals estimated by Equation (3.1). Three experiments are performed. In each of them, the packet headers are processed under a different sampling probability $p$,[4] thus producing three datasets of packet-sampled flow records. Confidence intervals for $n$ are calculated using Equation (3.1) with a significance level of 95 % and $r = 1$. Figure 3.5(a) shows that, as expected, the intervals are wider for lower $p$ values.

[4] As in Chapter 2, we convert the packet headers into flow records using YAF [89].

Figure 3.5(b) illustrates how the wider intervals make the decision about *unhealthy* traffic harder. The figure plots the confidence intervals when varying $p$ for an arbitrary pair $n$ and $n_h$ – note the logarithmic $x$-axis. Similar shapes would be obtained for other $n$ and $n_h$ as well. For each point in the figure, 10 random samples of $B(k|n,p)$ and $B(k|n_h,p)$ are taken and confidence intervals for $n$ and $n_h$ are calculated using Equation (3.1) with significance level of 95 % and $r = 10$ to improve visualization. In this example, our method would report an *unhealthy* service when $p > 0.002$ (approximately). For lower $p$ values, however, the intervals overlap, preventing the difference between $\hat{n}$ and $\hat{n}_h$ from being considered significant, even if only two thirds of the connections (*i.e.,* 40,000 out of 60,000) are *healthy.* Therefore, in order to compensate for the sampling process, the method is more conservative under lower $p$ values.

### 3.1.3 Prototype

In order to analyze cloud services in practice, we have integrated the method depicted in Figure 3.2 in an open source plug-in for NfSen [78]. NfSen is a Web interface for the NFDUMP collector. The decision of building our prototype on top of NfSen is motivated by the fact that (i) NFDUMP and NfSen are readily available on several Linux distributions; (ii) NFDUMP and NfSen are already widely deployed for other flow monitoring activities (*e.g.,* at our own network).

Figure 3.6 summarizes the architecture of our prototype, which follows the NfSen plug-in architecture [77]. NFDUMP collects flow records from several sources and passes on batches of data to our application periodically. Our prototype is embedded in a back-end NfSen plug-in. It receives the flow data and performs the steps illustrated in Figure 3.2 – *i.e.,* (i) flow records related to targeted services are filtered by means of parametrized IP address lists or the MaxMind GeoIP Organization [106] database; (ii) the flow data are normalized; and (iii) the *health indexes* are calculated for predefined services. The back-end NfSen plug-in saves partial information in binary files between execution rounds – *e.g.,* the connections that are not yet complete according to the heuristic described in Section 3.1.1. The calculated indexes are then archived in Round-Robin Databases (RRDs), together with other basic traffic statistics.

The periodicity that the prototype evaluates batches of flow records and outputs *health indexes* needs to be set in a practical deployment. Since we assume the method will be deployed next to the *existing* monitoring infrastructure, this parameter has to adhere to other flow export settings. The delay of collectors in saving and forwarding flow records to analysis applications determines the parameter – *e.g.,* NFDUMP uses 5 min by default. Hence, since the method is executed only when a new batch of flow records is available, the information about cloud services is updated every few minutes in a typical deployment.

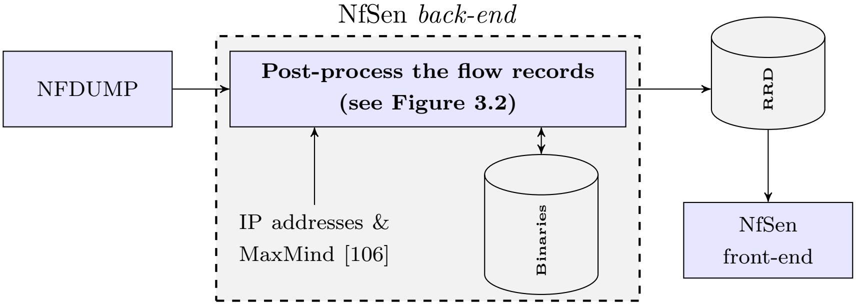

NfSen *back-end*
NFDUMP
**Post-process the flow records (see Figure 3.2)**
RRD
IP addresses & MaxMind [106]
Binaries
NfSen front-end

Figure 3.6: *Cloud Monitor* (based on the NfSen plug-in architecture [77]).

The status of cloud services can be graphically monitored using a *front-end* plug-in for NfSen. The design and implementation of this front-end plug-in will not be described in this thesis. Interested readers can find more information in [112], and download the source code of our prototype, named *Cloud Monitor*, from `http://www.simpleweb.org/wiki/Cloud_Monitoring`.

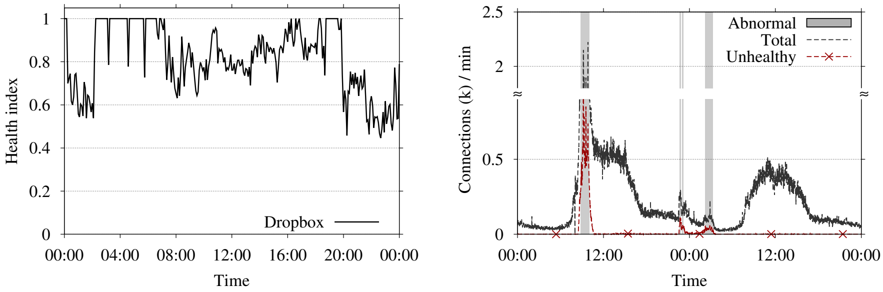
## 3.2 Case Study 1: Popular Cloud Services

This section validates our method using non-sampled flow data. Similar analysis will be performed in Section 3.3 using packet-sampled flows. Section 3.2.1 describes our dataset and methodology. The health of popular services is analyzed in Section 3.2.2. Finally, Section 3.2.3 summarizes the obtained results.

### 3.2.1 Dataset and Methodology

The prototype plug-in for NfSen described in the previous section is deployed at our university network. All flow data exported by edge routers (non-sampled NetFlow v9) are processed, and statistics of the aggregated records are archived in 1-min bins. This section evaluates the data collected by our prototype in the first 10 weeks of 2013.

Four organizations are analyzed: Dropbox, Twitter, Google and Facebook. These organizations have been selected because their services are highly popular in our network. We have documented all IP addresses serving Dropbox, and the MaxMind databases are used to filter flows to/from Twitter, Google and Facebook. Note that some content related to these organizations is hosted by third-parties (*e.g.,* Facebook's static content is hosted by Akamai). Our results do not include this traffic.



(a) Blacklisted clients: Mar 7, 2013

(b) Jan 10–11, 2013

Figure 3.7: Monitoring Dropbox within the UT network.

The most serious performance anomalies in this dataset have been analyzed manually and will be discussed in the following. Furthermore, the official channels in which Dropbox [47], Twitter [142] and Google [70] report the status of their services are used to understand how our prototype reacts to different performance anomalies. Facebook is left out of this last analysis because no historic information about its services could be found.

### 3.2.2 Health Analysis

Our results show an index normally close to 1, with fluctuations during the periods in which few users are interacting with the services – *i.e.,* late in the nights. The fluctuations happen because any connection failures have a significant impact on the index, since the total number of connections in these periods is very low. However, several abnormal situations can also be noticed during business hours. Figure 3.7(a) depicts a 1-day example of the *health index* for Dropbox. We can see that the index constantly fluctuates in our network, indicating that many clients are unable to access the service. By manual inspection, we have identified that the fluctuations are caused by abandoned Dropbox clients, which are sometimes blacklisted by the university firewall. Such clients keep trying to contact Dropbox servers, increasing the number of *unhealthy* TCP connections. Even though the root cause of the problem is in our own network, the prototype correctly points to an *unhealthy* service.

After removing blacklisted clients, few abnormal intervals remain. The index for the selected organizations is lower than 0.8 in 524 min (0.52 % of the 1-min bins) and lower than 0.7 in 285 min (0.28 % of the 1-min bins), for instance. For validating these results, all cases in which the index stays below 0.7 for more than 5 consecutive minutes have been evaluated. These thresholds have been chosen in order to limit the time consuming manual analysis to the most severe cases. In total, 8 (non-overlapping) intervals with a total duration of 201 min have been manually analyzed, distributed among Dropbox (7) and Twitter (1).

Dropbox reported problems twice in the period. Abnormal intervals (5 in total) appear in our dataset in both cases. Figure 3.7(b) depicts examples seen on Jan 10–11: three periods in which the amount of *unhealthy* traffic to Dropbox increases can be seen in the figure (see gray regions). By inspecting the remaining 2 abnormal intervals, which do not coincide with official reports from Dropbox, we conclude that only a non-essential part of the service has been affected,[5] suggesting that the seven most serious cases reported by our prototype are all correct alerts.

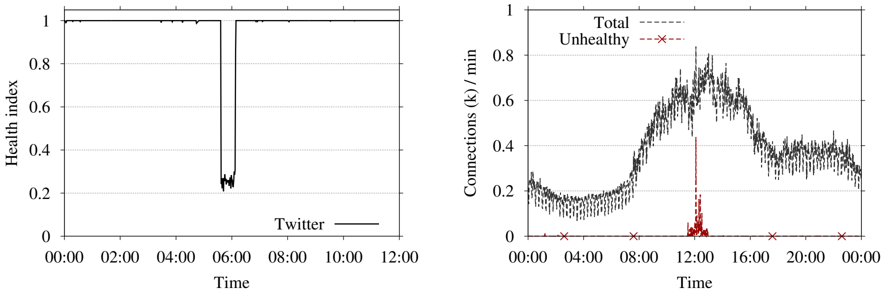
The abnormal intervals in Figure 3.7(b) also illustrate a consequence of ignoring application layer protocols while processing the flow records. In all three periods marked in gray in the figure, we can see that the total number of TCP connections to Dropbox increases significantly. A close look into the flows reveals that not only the number of *unhealthy* connections increases, but also the number of *healthy* connections, suggesting errors at the application layer. Since the proportion of *unhealthy* connections is high, the problems are detectable, as depicted in the figure. However, this example clearly demonstrates that our metric is an approximation – *i.e.,* application layer protocols need to be taken into account for identifying *all unhealthy* traffic.

The abnormal interval related to Twitter (see Figure 3.8(a)) does not coincide with official reports. However, a simultaneous increase on the number of *unhealthy* TCP connections to several destinations can be seen in our data. Hence, our prototype seems to correctly point to a problem in our network, although the root cause is unclear in this case.

Twitter officially reported problems 6 times in the period. None of these reports passes the threshold set to this experiment. However, two of them are clearly visible in our dataset. Figure 3.8(b) shows an example: the interval with an increased number of *unhealthy* TCP connections coincides with a Twitter's announcement of site problems – note the decrease also in the total number of connections. Twitter also reported site problems affecting part of its users on Jan 17 and Jan 31. These 2 cases are not visible in our data, and it is not clear whether our users experienced problems. The remaining 2 cases involved a malfunction and a bug of specific site functionalities, which are not expected to be detected by our prototype, since they are application layer errors.

Finally, Google reported 5 problems in the period, including longer response times and errors with attachments on Gmail. None of the issues prevented users

[5] The problem has affected only servers that collect run-time statistics from Dropbox clients. Details of the Dropbox service will be provided in Chapter 4.



(a) 12 hours on Feb 26, 2013

(b) Feb 27, 2013

Figure 3.8: Monitoring Twitter within the UT network.

from accessing the services. The method proposed in this chapter, as expected, does not report these problems.

### 3.2.3 Summary

The analyzed companies reported 13 incidents in the studied 10-week interval. Among those, 7 cases are out of our target, since they involve application layer errors. Moreover, 2 incidents might not have affected our network. The remaining 4 cases are detectable in our data, even though the services were still partially accessible. Our method also identified anomalies that either were not officially reported or affected only our own network. Overall, the case study shows the applicability of flow measurements to monitor cloud services.

Note that, while we have analyzed the drops in the *health index* manually in this section, a more user friendly alert system requires an automatic post-processing of the output of our prototype. The design of such a system is out of the scope of this chapter.

## 3.3 Case Study 2: the WikiLeaks Cablegate

We now show the applicability of our method using packet-sampled flow data. The case study in this section is the series of cyber-demonstrations promoted by a group of *hacktivists* named *Anonymous* after the release of U.S. embassy cables by the WikiLeaks in 2010. As a reaction to those leaks, several companies retracted their business support to WikiLeaks. *Anonymous* reacted to this boycott by coordinating denial-of-service attacks against MasterCard, PayPal, among others. Some of the targets were reported to be inaccessible during the cyber-demonstrations [105], although this has never been confirmed.

*Anonymous* used a tool named `LOIC` against the targets. `LOIC` has been originally developed for stress testing on servers and, as such, does not implement any sophisticated attack method. The tool uses standard libraries for establishing TCP connections to a remote server and sending user-defined payloads. Hence, `LOIC` generates regular traffic at the transport layer. For these cyber-demonstrations, the tool was extended to obtain configuration parameters from IRC servers – *i.e., hacktivists* voluntarily joined a botnet.

Section 3.3.1 describes our dataset and methodology. Section 3.3.2 evaluates the health of several targets. Finally, Section 3.3.3 summarizes the results.

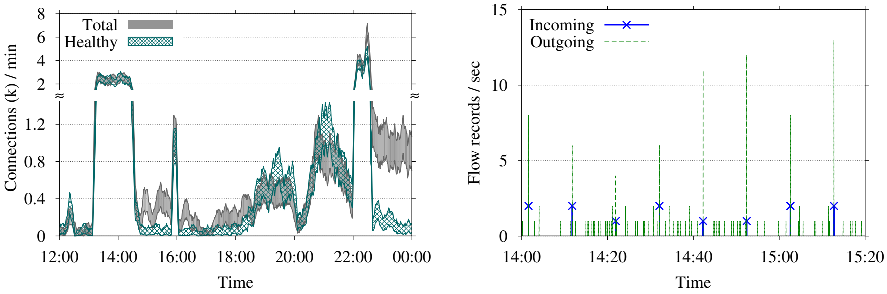
### 3.3.1 Dataset and Methodology

Packet-sampled NetFlow records (with sampling probability $p = 0.01$) collected at an international backbone are used in this analysis. The following services involved in the WikiLeaks Cablegate are analyzed: PayPal's payment services, MasterCard's website, Moneybookers' website, the website of Senator Liberman and PayPal's blog. We give particular attention to the case of PayPal, since the data indicate that active attack containment measures have been taken. Our decision to focus on these targets is motivated exclusively by the amount of traffic to each target in our dataset. Several other organizations and politicians have been targeted by *Anonymous*. A complete description of the *WikiLeaks Cablegate*, including all targets and *Anonymous*' motivations for attacking each specific website, is presented in [105, 120].

The method in Section 3.1.2 estimates the number of TCP connections as well as the number of *healthy* connections for each target. Confidence intervals for a significance level of 95 % and parameter $r = 10$ (see Equation (3.1)) are calculated for time bins of 1 min – *i.e.,* all results are calculated considering 10 consecutive measurements to reduce the effects of outliers and to alert only for the most serious problems. In order to provide a better insight into our method to sampled flows, this section shows the computed confidence intervals for the estimated number of connections, instead of the *health index.*

### 3.3.2 Health Analysis

PayPal's payment service was targeted by *hacktivists* on Dec 9, 2010. Figure 3.9(a) shows the obtained intervals for the number of TCP connections to PayPal's service. Whenever the confidence intervals do not overlap, a possible degradation of the service has occurred. We observe that the intervals do not overlap for more than one hour around 15:00 and for almost two hours



(a) Health analysis

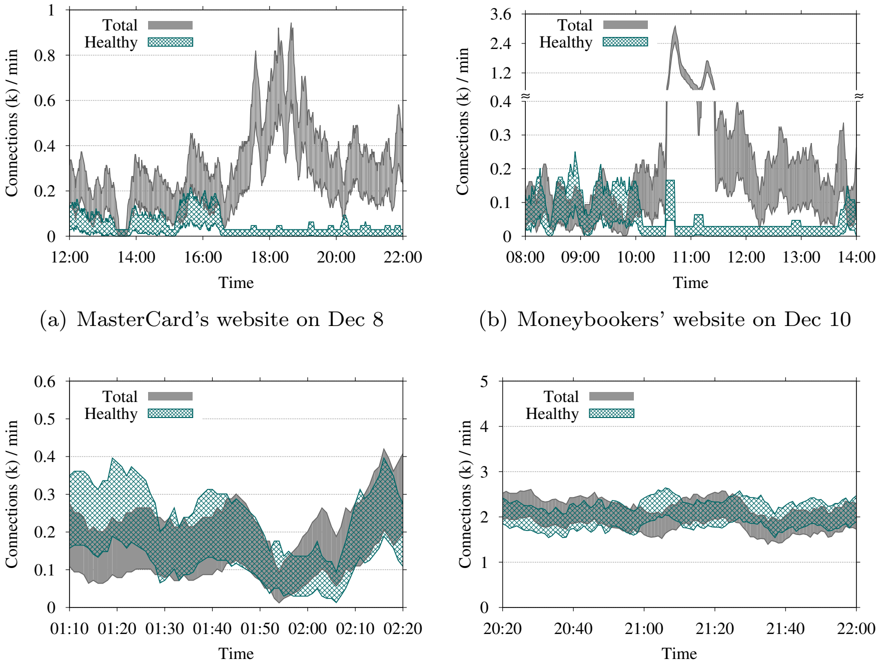
(b) Firewall regulating a *hacktivist*

Figure 3.9: PayPal's service on Dec 9. Note the discontinuity in the $y$-axis.

after 22:30, which is an evidence of *unhealthy* traffic and, possibly, performance anomalies. For other periods, there are no clear signs that the cyber-demonstrations caused any damage or health degradation.

For the periods in which the confidence intervals do not overlap, there are indications that *hacktivists* were being blocked by a firewall. The analysis of `LOIC` source code reveals that the tool implements a very inefficient stress testing strategy. `LOIC` submits a new request only if a response from the servers is received or a timeout interval is elapsed – *i.e.,* targets could easily slow down the attack by deploying a firewall close to their servers. Figure 3.9(b) suggests that PayPal has adopted such a solution. The figure shows the number of raw flow records from/to one potential *hacktivist*: incoming traffic occurred periodically, which is a typical behavior of gray-listing firewalls. The way `LOIC` is implemented explains the peaks of outgoing records every time servers started responding again. Since the service can be considered *unhealthy* from the point of view of this *hacktivist*, our method produces correct results also for this case.

Figure 3.10 depicts the analysis for other targets. Figure 3.10(a) shows two clearly different phases regarding MasterCard's website on Dec 8: until around 17:00, the number of *healthy* TCP connections was consistently lower than the total number of TCP connections, although the difference cannot be considered significant in all time intervals; after that, there were almost no responses from servers anymore. During the first phase, it is likely that the service was experiencing problems, but it was still able to handle a portion of the traffic. In the second phase, *hacktivists* were likely blocked, as in the case of PayPal. Note that this backbone normally does not transport significant amount of requests to MasterCard and, therefore, the observed traffic may have been generated mostly by *hacktivists*. Overall, our method points to an *unhealthy*



(a) MasterCard's website on Dec 8

(b) Moneybookers' website on Dec 10

(c) Senator Lieberman's website on Dec 8

(d) PayPal's Blog on Dec 10

Figure 3.10: Several *Anonymous*' targets. Note the discontinuity in the $y$-axis.

service in several intervals, even though most of the client requests seem to come from possible *hacktivists*.

The same comments are valid for the website of Moneybookers (Figure 3.10(b)). However, the website was *healthy* almost all the time, until it stopped handling *hacktivists*' requests completely (around 10:30). Finally, Figure 3.10(c) and Figure 3.10(d) show that both the website of U.S. Senator Lieberman and PayPal's blog were not affected by the cyber-demonstrations during the analyzed time intervals.

### 3.3.3 Summary

Our method identifies a large amount of *unhealthy* traffic to targeted services during the analyzed periods. However, most *unhealthy* traffic seems to be a consequence of firewalls blocking *hacktivists*. The predictable behavior of `LOIC` makes it easy to identify `LOIC` traffic in flow datasets. By manually analyzing the data, it can be concluded that most TCP connections in the attack days are from few sources producing an abnormally high number of requests that match with `LOIC` behavior. This pattern is, on the other hand, not observed on other days in the same network. These results suggest that, although *Anonymous* claimed that some targets were off-line [105], targeted services have been only marginally affected by these cyber-demonstrations.

## 3.4 Lessons Learned

This section discusses some lessons learned from the case studies. These lessons are important because they illustrate general limitations of using NetFlow to monitor cloud services and provide us with guidelines for extending our method in the next chapter.

**The need for better traffic identification methods.**

The method evaluated in this chapter filters traffic based on IP addresses and IP owners. Our case studies show that the method performs well if flow records are filtered properly. However, this methodology to filter traffic has proven inconvenient and ineffective. Indeed, some *Anonymous*' targets (*e.g.,* Visa) are not analyzed in our second case study because the traffic from/to those services could not be identified in our dataset. The problem happens because cloud providers host several services simultaneously. Neither IP addresses nor popular methods to indicate IP owners – *e.g.,* the MaxMind databases – provide hints on how customers are allocated in the providers' address space.

The first lesson learned, therefore, is the need for methods to identify flows generated by an application. In particular, the limited information normally exported by NetFlow makes it hard to isolate the traffic of cloud services in flow datasets. Part II will overcome this limitation by augmenting flow measurements with other information that can be passively observed in the network as well.

**Can application layer semantics be ignored?**

Motivated by our goal of building a generic monitoring system, application layer protocols are not analyzed in this chapter. Our results show that our proposed method is informative to reveal the most severe availability problems, in which clients cannot establish communication with providers. However, both case studies also demonstrate that it is very common that servers are still able to handle part of users' requests when suffering performance degradation. In such situations, a mix of *healthy* traffic, *unhealthy* traffic at the transport layer, and *unhealthy* traffic at the application layer can be observed in the network – *e.g.,* see the discussion about Figure 3.7(b) in Section 3.2.2.

Hence, the second lesson learned from the case studies is that application-specific knowledge has to be taken into account as well, if the precise identification of *all* availability problems is necessary. Such advanced monitoring requires flow exporters that can measure and export customized information per application. Part II will extend our method by relying on a specialized flow exporter that collects more than 100 metrics about the flows.

**Other performance problems are common.**

This chapter introduces a simple metric that helps to reveal the status of cloud services based on NetFlow data. As illustrated in our first case study (see Section 3.2), however, other equally important performance problems might be even more frequent and cannot be monitored by means of this metric. The next chapter will extend the analysis to other performance problems, by restricting our scope to cloud storage services.

## 3.5 Related Work

This chapter discusses how to monitor problems in cloud services. Several works rely on active measurements to analyze the performance of services or to benchmark cloud providers [93, 99, 101]. Other works [108, 109] focus on measuring the performance of the infrastructure providing the services. In some cases [87, 147], a particular cloud service or provider is analyzed in detail. Our work differs from those in several aspects. Firstly, the increasing variety of client platforms – *e.g.,* mobile devices – motivates our decision to monitor at the network. Secondly, we do not focus on a specific service, but instead, we propose a generic method applicable to a variety of cloud services. Lastly, we take a passive approach to cloud monitoring because the active alternatives lack the ability to capture the impact of problems on actual end users. Active approaches are, however, complementary to ours, since they provide another view of the services. For example, in contrast to active methods, our method can only identify problems in a cloud service if end users are interacting with it.

The use of NetFlow for measuring the availability of a remote service is central to our method. A similar approach is proposed in [68, 128] to assist operators in identifying connectivity problems and outages. Incoming/outgoing flow records are matched and alerts are triggered when the number of records without a match increases. Our work differs from those in two aspects: firstly, when dealing with non-sampled data, our method uses all information available on flow records to determine whether TCP connections are *healthy* or not. Secondly, [68, 128] assume non-sampled flow data, while our method also provides support to packet-sampled flows.

When dealing with non-sampled data, we rely on a technique for aggregating flow records that is based on [130]. The authors of [103] use an opposite approach: timeout parameters of flow exporters are tuned for approximating flow records to TCP connections in a better way. Our work diverges strongly from [103], since we normalize the flow data, thus removing the effects of timeout parameters. For post-processing packet-sampled flows, we follow the theoretical framework presented in [52]. The authors of [52] show how to estimate several properties of the original data stream, such as the flow length distribution, using packet-sampled flow records only. We use a similar reasoning to estimate the distributions of the number of TCP connections per health state.

Finally, this chapter contributes with a practical implementation of our method as a plug-in for NfSen. Several works propose extensions for NfSen [15, 86, 121]. Among those works, Nfsight [15] has goals closest to ours. Nfsight also implements a heuristic to post-process flow records, identifying originators and responders of TCP connections and calculating connection statistics. However, in contrast to our method, Nfsight focuses on intrusion detection.

## 3.6 Conclusions

The main contribution of this chapter is a simple method for monitoring the availability of cloud services that relies solely on NetFlow. Our method has the distinct characteristic of being able to cope with both non-sampled and packet-sampled flow data. Hence, it explicitly targets high-speed networks. The method first isolates traffic related to a service and normalizes the data to remove the effects of parameter settings of flow exporters. Then, it estimates the fraction of *healthy* traffic to indicate the status of the monitored service.

Results of two case studies showed that our method can help customers to monitor performance of generic cloud services, by means of flow data that are normally available in current networks. Our first case study revealed problems in popular services, even if we measured in a somehow short time interval, thus providing evidences of availability problems in cloud providers. The applicability of our method was further confirmed by studying packet-sampled flow data collected during the *WikiLeaks Cablegate*. We concluded, for example, that targeted services were only marginally affected by the cyber-demonstrations, thus contradicting *Anonymous*' claims.

The experience acquired with these two case studies highlighted limiting factors for the use of NetFlow to monitor cloud services. These factors include the difficulties for filtering traffic and the need for more information in flow records. The lessons learned will guide the definition of our methodology to monitor performance of cloud storage services in the next chapter.

Determining the health of a remote service solely from flow measurements is challenging. The method proposed in this chapter is by no means a comprehensive solution to all monitoring needs of cloud customers. However, by focusing on serious performance problems and by relying on flows exported by a widely deployed technology, it delivers an essential and immediate first layer of monitoring to cloud customers, while being applicable to a wide-range of services and scalable to high-speed networks.

Finally, our prototype has been integrated into NfSen and is available to the public at `http://www.simpleweb.org/wiki/Cloud_Monitoring`.

# Part II

# Cloud Storage Services

CHAPTER 4

# Dropbox Usage and Performance

Cloud storage services allow to synchronize local folders with servers in the cloud. They have gained popularity, with companies offering remote storage for free or accessible prices. More and more people are being attracted by these offers, saving personal files, synchronizing devices and sharing content with great simplicity. This high public interest pushed various providers to enter the cloud storage market. Services like Dropbox and SkyDrive are becoming pervasive in people's routine. Dropbox, the best known offer by the time of writing in 2013, has been active since since 2007 and is used by over 175 million users, who upload more than 1 billion files on daily basis [48]. Considering its continuous growth in popularity, it is to be expected that cloud storage will soon be one of the top applications generating Internet traffic.

Although cloud storage has attracted massive attention, very little is known about typical workload and performance bottlenecks of such services. This understanding is essential, firstly, for organizations interested in outsourcing file storage systems, both to guide their decision on a possible migration and to help monitor already migrated services. Secondly, as social and privacy issues impel the appearance of alternative providers, such knowledge becomes more and more important for the design of new, well-performing services.

The main goal of this chapter is to provide a comprehensive characterization of cloud storage services. We analyze flow measurements collected from two university campuses and two Points of Presence (POPs) in a large ISPs for 42 consecutive days. We first extend our general methodology to monitor performance of cloud services, devising methods for both isolating cloud storage traffic and calculating application-specific performance metrics. We then focus on Dropbox, which we show to be the most widely-used cloud storage service in our datasets. Dropbox already accounts for about 100 GB of daily traffic in one of the monitored networks – *i.e.,* 4 % of the total traffic or around one third of the YouTube traffic in the same network. We focus first on the service performance characterization, highlighting possible bottlenecks and suggesting countermeasures. Then, we detail user habits, thus providing an extensive characterization of the workload the system has to face.

To be best of our knowledge, we are the first to provide an analysis of Dropbox usage on the Internet. The authors of [87] compare Dropbox, Mozy, Carbonite and CrashPlan, but only a simplistic active experiment is provided to assess them. The authors of [148] note the existence of performance bottlenecks in Dropbox, but the root causes of the problem are not clearly identified, as we will do in this chapter. In [113], the possibility of unauthorized data access and the security implications of storing data in Dropbox are analyzed. We follow a similar methodology to dissect the Dropbox proprietary protocols in Section 4.1, but focus on a completely different problem.

Considering storage systems in general, [81, 82] study security and privacy implications of the deployment of data deduplication – the mechanism in place in Dropbox for avoiding the storage of duplicate data. Some works explicitly consider cloud architecture designs [98], while others present a comparison among different cloud providers [101, 147]. By running benchmarks, these works focus on the achieved performance of different providers, but focusing only on server infrastructure, missing the characterization of both the typical usage of a cloud service and the impact of user behavior on cloud storage applications. Similarly, [12] presents a performance analysis of the AWS in general, but does not provide insights into cloud storage. Finally, several works characterize popular services, such as social networks [67, 111] or YouTube [23, 60]. Our work goes in a similar direction, shedding light on Dropbox and possibly other related systems.

This chapter will show that cloud storage applications are data hungry and user behavior deeply affects their network requirements. Our results are useful for the research community, ISPs and companies interested in outsourcing, to understand the impact of a massive adoption of such solutions. Similarly, our analysis of the Dropbox performance is a reference for engineers designing protocols and provisioning data centers for similar services, with valuable lessons about bottlenecks introduced by some design decisions.

The remainder of this chapter is organized as follows. Section 4.1 describes the methodology we use to reverse-engineer the Dropbox protocols, and provides a short description of Dropbox architecture and its client functioning. Section 4.2 describes our data collection. Section 4.3 studies the traffic volume generated by several cloud storage services, evaluating the popularity of well-known providers in the analyzed networks and characterizing the overall workload of cloud storage. Section 4.4 presents a characterization of Dropbox performance and possible bottlenecks of the system. User habits and the generated workload are presented in Section 4.5. Finally, Section 4.6 concludes this chapter and lists our main contributions.

Table 4.1: Domain names used by different Dropbox services. Numeric suffixes are replaced by ‘X’.

| sub-domain | Data center | Description |
|---|---|---|
| client-lb/clientX | Dropbox | Meta-data |
| notifyX | Dropbox | Notifications |
| api | Dropbox | API control |
| www | Dropbox | Web servers |
| d | Dropbox | Event logs |
| dl | Amazon | Direct links |
| dl-clientX | Amazon | Client storage |
| dl-debugX | Amazon | Back-traces |
| dl-web | Amazon | Web storage |
| api-content | Amazon | API Storage |

## 4.1 Dropbox Overview

### 4.1.1 The Dropbox Client

The Dropbox native client is implemented mostly in Python, using third-party libraries such as *librsync*. The application is available for Microsoft Windows, Apple OS X and Linux.[1] The basic object in the system is a *chunk* of data with size of up to 4 MB. Files larger than that are split into several chunks, each treated as an independent object. Each chunk is identified by a SHA256 hash value, which is part of the meta-data descriptions of files. As Chapter 5 will explain in details, Dropbox reduces the amount of exchanged data by using delta encoding when transmitting chunks. It also keeps locally in each device a database of meta-data information (updated via incremental updates) and compresses chunks before submitting them. In addition, the client offers the user the ability to control the maximum download and upload speed.

Two major components can be identified in the Dropbox architecture: the *control* and the *storage* servers. The former are under direct control of Dropbox Inc., while Amazon EC2 and Amazon S3 are used as storage servers. In both cases, sub-domains of `dropbox.com` are used for identifying the parts of the service offering a specific functionality (see Table 4.1). HTTPS is used to access all services, except the *notification* service, which runs over HTTP.

[1] Mobile device applications (*i.e.,* running on iOS and Android) access Dropbox on demand using Application Programming Interfaces (APIs); those are not considered here.

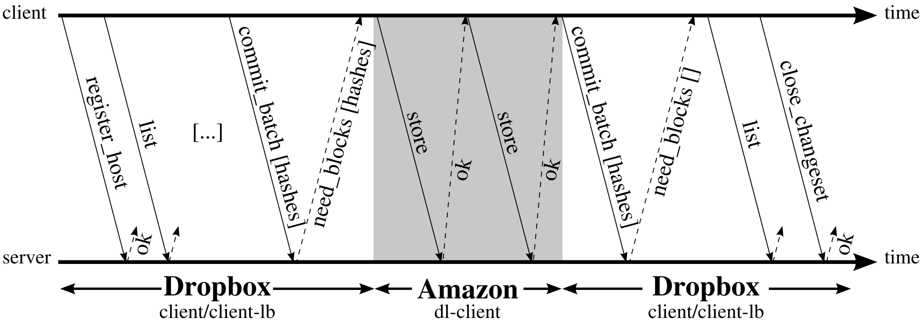


Figure 4.1: An example of the Dropbox protocol.

### 4.1.2 Understanding Dropbox Internals

In order to calculate performance metrics that are meaningful at the application layer and characterize the usage of the service from passive measurements, we first need to gain an understanding of the Dropbox client protocol. We have performed several active experiments to observe what information is exchanged after a particular operation. For instance, among others, we have documented the traffic generated when adding or removing files on local folders, when downloading new files and when creating new folders. During these experiments, Dropbox was distributing its client version 1.2.52 as the stable version.

Since most client communication is encrypted with Transport Layer Security (TLS) and very little public information is released by Dropbox, we set up a testbed equipped with an *intercept proxy.* We use a `Squid` proxy server, extended with the module `SSL-bump` [124], to terminate TLS/SSL connections and save decrypted traffic flows. The Dropbox client needs to be instructed to trust the self-signed proxy certificate. Because trusted certificate authorities are hard-coded in the Dropbox client, we use debugging tools (*e.g.,* `GDB`) to manipulate the memory heap in our testbed. The memory area where trusted certificate authorities are stored can be modified at run-time to replace a trusted certificate by the self-signed one of the proxy server. By means of this setup, we were able to observe and to understand the Dropbox client communication.

Figure 4.1 illustrates the messages we observe while committing a batch of chunks. After registering with the Dropbox control center via a `clientX.dropbox.com` server, the *list* command retrieves meta-data updates. As soon as new files are locally added, a *commit_batch* command (on `client-lb.dropbox.com`) submits meta-data information. This can trigger *store* operations, performed directly with Amazon servers (on dl-clientX.dropbox.com). Each chunk *store* operation is acknowledged by one *OK* message. As we will see in Section 4.4, this acknowledgment mechanism might originate performance bottlenecks. Finally, as chunks are successfully submitted, the client exchanges messages with the central Dropbox servers (on client-lb.dropbox.com) to conclude the transactions. Note that messages committing transactions might occur in parallel with newer *store* operations.

A complete description of the Dropbox protocols is outside the scope of this thesis, since protocols are likely to change in the future. We, however, exploit this knowledge both (i) to filter the passively observed TCP flows with the likely commands executed by the client; and (ii) to derive meaningful performance metrics from network flows passively collected from operational networks, even if we have no access to the content of the (encrypted) connections. In the following, we describe the protocols used to exchange data with the Dropbox control servers and with the storage servers at Amazon.

### 4.1.3 Client Control Flows

The Dropbox client exchanges control information mostly with servers managed directly by Dropbox Inc. Three sub-groups of control servers can be identified: (i) notification servers; (ii) meta-data administration servers; and (iii) system-log servers. System-log servers collect run-time information about the clients, including exception back-traces (via Amazon, on dl-debug.dropbox.com), and other event logs possibly useful for system optimization (on d.dropbox.com). Since flows to those servers are not directly related to the usage of the system and do not carry much data, they are not considered further. In the following, we describe the key TCP flows to the meta-data and notification servers.

#### Notification Protocol

The Dropbox client keeps open a TCP connection to a notification server (notifyX.dropbox.com), used for receiving information about changes performed elsewhere. In contrast to other traffic, notification connections are *not encrypted*. Delayed HTTP responses are used to implement a push mechanism: a notification request is sent by the local client asking for possible changes; the server response is received periodically about 60 s later in case of no change; after receiving it, the client immediately sends a new request. Changes on the central storage are instead advertised as soon as they are performed.

Each device linked to Dropbox has a unique identifier (*host_int*). Unique numeric identifiers (called *namespaces*) are also used for each shared folder. The client identifier is sent in notification requests, together with the most updated list of namespaces. Devices and shared folders can, therefore, be identified in network traces by passively watching notification flows, thus representing a privacy threat. Finally, different devices belonging to a single user can be inferred as well, by comparing namespace lists.

#### Meta-Data Information Protocol

Authentication and meta-data administration are handled by a separate set of servers, (`client-lb.dropbox.com` and/or `clientX.dropbox.com`). Typically, synchronization transactions start with client messages to meta-data servers, followed by a batch of either *store* or *retrieve* operations through Amazon's servers. As data chunks are successfully exchanged, the client sends messages to meta-data servers to conclude the transactions (see Figure 4.1). Due to an aggressive TCP connection timeout handling, several short TLS connections to meta-data servers can be observed during this procedure.

Server responses to client messages can include general control parameters. For instance, our experiments in the testbed reveal that the current version of the protocol limits the number of chunks to be transferred to at most 100 per transaction. That is, if more than 100 chunks need to be exchanged, the operation is split into several batches, each of at most 100 chunks. Such parameters shape the traffic produced by the client, as it will be analyzed in Section 4.4.

### 4.1.4 Data Storage Flows

As illustrated in Figure 4.1, all *store* and *retrieve* operations are handled by virtual machines in Amazon EC2. More than 500 distinct domain names (`dl-clientX.dropbox.com`) point to Amazon servers. A subset of those aliases are sent to clients regularly. Clients rotate in the received lists when executing storage operations, distributing the workload.

Typically, storage flows carry either *store* commands or *retrieve* commands. This permits storage flows to be divided into two groups by checking the amount of data downloaded and uploaded in each flow. By means of the data collected in our test environment, we documented the overhead of *store* and *retrieve* commands and derived a method for labeling the flows. Furthermore, we identified a direct relationship between the number of TCP segments with the `PSH` flag set in storage flows and the number of transported chunks. For the sake of brevity, we present more details about our methodology in Appendix B, along with some results validating that the models built in our test environment represent the traffic generated by real users satisfactorily. We use this knowledge in the next sections, in combination with a specialized flow exporter, for characterizing the system performance and workload.

### 4.1.5 Web Interface and Other Protocols

Content stored in Dropbox can also be accessed through Web interfaces. A separate set of domain names are used to identify the different services and can thus be exploited to distinguish the performed operations. For example, Uniform Resource Locators (URLs) containing `dl-web.dropbox.com` are used when downloading private content from user accounts. The domain `dl.dropbox.com` provides public direct links to shared files.

In addition, a protocol for synchronizing machines in the same LAN is available in the Dropbox client. The protocol operation is twofold: (i) UDP broadcasts are sent out containing the namespaces of a device – other devices can, therefore, form a list of possible pairs for future synchronizations; (ii) peer-to-peer connections are established when a notification of changes in previously announced namespaces is received. Users with several folders in common can make use of the protocol, with a positive impact on network traffic. The percentage of users potentially profiting from this protocol will be analyzed in Section 4.5.

## 4.2 Datasets and Methodology

### 4.2.1 Methodology

We rely on passive flow measurements to analyze the Dropbox traffic in operational networks. In contrast to the first part of the thesis, we use a specialized flow exporter in this chapter, since we aim at an advanced analysis of Dropbox. We rely on Tstat [59] to collect data. Tstat is a passive sniffer and flow exporter that monitors each TCP connection, exposing information about more than 100 metrics,[2] including client and server IP addresses, the amount of exchanged data, retransmitted segments, RTT and the number of TCP segments that have each TCP flag set [107].

Specifically targeting this analysis, we implement several additional features. Firstly, given that Dropbox relies on HTTPS, we extract the TLS/SSL certificates offered by the server by means of Deep Packet Inspection (DPI), and export the information as an extra field in the flows. Our analysis shows that the string `*.dropbox.com` is used to sign all communications with the servers. Secondly, we augment the exposed information by labeling server IP addresses with the original Fully Qualified Domain Name (FQDN) the client requested to the DNS server [13]. Both extensions are key to reveal information on the server that is being contacted (see Table 4.1) and allow us to identify each specific

[2] See `http://tstat.tlc.polito.it` for details.

Table 4.2: Datasets overview (total traffic over 42 days).

| Name | Type | IP Addresses | Volume (GB) |
|---|---|---:|---:|
| Campus 1 | Wired | 400 | 5,320 |
| Campus 2 | Wired/Wireless | 2,528 | 55,054 |
| Home 1 | FTTH/ADSL | 18,785 | 509,909 |
| Home 2 | ADSL | 13,723 | 301,448 |

Dropbox functionality from the passively collected flows. Thirdly, Tstat is instructed to expose the list of device identifiers (*host_int*) and folder namespaces exchanged with the notification servers. Finally, using the knowledge presented in Section 4.1 and Appendix B, we calculate extra performance metrics specific to Dropbox, as we will describe below.

### 4.2.2 Datasets

We installed Tstat at 4 vantage points in 2 European countries and collected data from Mar 24, 2012 to May 5, 2012. This setup provided a valuable pool of datasets, allowing us to analyze the use of cloud storage in different environments, which vary in both the access technology and the typical user habits. Table 4.2 summarizes our datasets, showing, for each vantage point, the access technologies present in the monitored network, the number of unique client IP addresses, and the total amount of data observed during the whole period.

The datasets labeled *Home 1* and *Home 2* consist of ADSL and Fiber to the Home (FTTH) customers of a nation-wide ISP. Customers are provided with static IP addresses, but they might use Wi-Fi routers at home to share the connection. *Campus 1* and *Campus 2* are collected in academic environments: *Campus 1* monitors wired workstations in research and administrative offices of the Computer Science Department of a European university. *Campus 2* accounts for all traffic at the border routers of a second university, including campus-wide wireless access points and student houses. NAT and HTTP proxy-ing are very common in this latter scenario, and DNS traffic is not exposed to the probe. For privacy reasons, we cannot provide further information about the networks where the data have been collected. Similarly, to protect users' privacy, our probes export only flows and the extra fields described in the previous section. Payload data are discarded directly in the probe.

Dropbox overall traffic in our datasets is summarized in Table 4.3, where we can see the number of flows, data volume, and devices linked to Dropbox in the monitored networks. Our datasets account for more than 11,000 Dropbox devices, uniquely identified by their device identifiers (*host_int*). The traffic generated by the Web interface and by Dropbox public APIs is also included. In

Table 4.3: Total Dropbox traffic in the datasets.

| Name | Flows | Volume (GB) | Dropbox devices |
|---|---|---|---|
| Campus 1 | 167,189 | 146 | 283 |
| Campus 2 | 1,902,824 | 1,814 | 6,609 |
| Home 1 | 1,438,369 | 1,153 | 3,350 |
| Home 2 | 693,086 | 506 | 1,313 |
| Total | 4,204,666 | 3,624 | 11,561 |

total, more than 3.5 TB were exchanged with Dropbox servers during our 42-day capture. To evaluate the performance implications of a protocol improvement announced by Dropbox, a second dataset was collected in *Campus 1* in Jun/Jul 2012, after the release of Dropbox version 1.4.0.

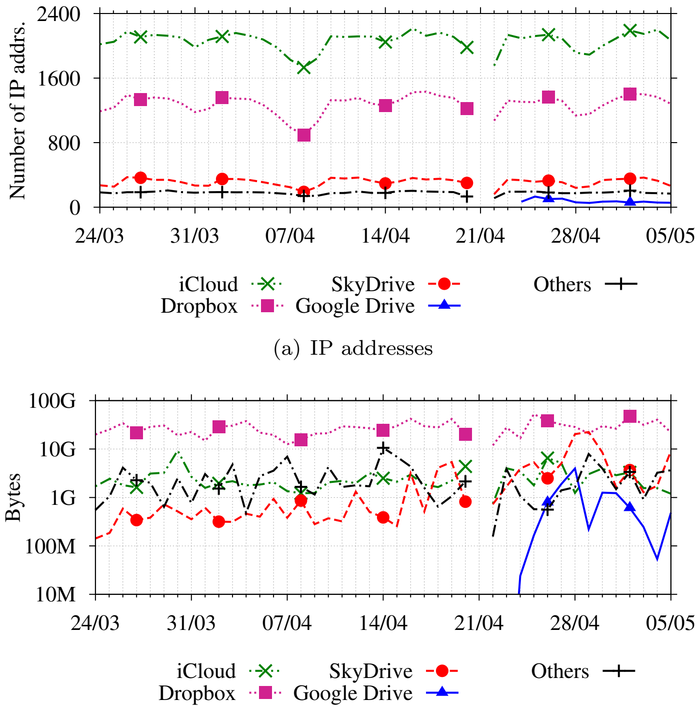
## 4.3 Popularity of Different Storage Providers

We compare the popularity of storage services and characterize the overall traffic volume to cloud storage in our datasets. We explicitly consider the following services: *Dropbox*, *Google Drive*, *Apple iCloud* and *Microsoft SkyDrive*. Other services (*e.g., Amazon Cloud Drive*, *Wuala* and *SugarSync*) are aggregated into the *Others* group. We rely on both the TLS server name and DNS FQDN to classify flows as belonging to each service.

We first study the popularity of the different services in terms of unique clients. We use the *Home 1* dataset because IP addresses are statically assigned to households and, therefore, are a reliable estimation of the number of installations. Figure 4.2(a) reports[3] the number of distinct IP addresses that contacted a storage service at least once in a given day. iCloud is the most accessed service, with about 2,100 households (11.1 %), showing the high popularity of Apple devices. Dropbox comes second, with about 1,300 households (6.9 %). Other services are much less popular (*e.g.,* 1.7 % for SkyDrive). Interestingly, Google Drive appears immediately on the day of its launch (April 24th, 2012). More importantly, these results can be directly correlated to statistics of search queries published by the Google Trends [74], suggesting that the usage of cloud storage services in our datasets is in-line with the global trend.

Figure 4.2(b) reports the total data volume for each service in *Home 1*. Dropbox tops all other services by one order of magnitude (note the logarithmic y-scale), with more than 20 GB of data exchanged every day. iCloud volume is limited despite the higher number of devices, because the service does not

[3] A probe outage is visible on April 21, 2012.



(a) IP addresses

(b) Data volume

Figure 4.2: Popularity of cloud storage in *Home 1* (in 2012).

allow users to synchronize arbitrary files, but only pre-define files, such as users' contacts, calendar etc. SkyDrive and Google Drive show a sudden increase in volume after their public launch in April. Plots in both Figure 4.2(a) and Figure 4.2(b) are relatively stable during the 42 days of observations, which suggests that usage and network requirements of cloud storage services are relatively predictable. Such findings are important for the provision of resource for similar services, and will be evaluated further in Section 4.5.

Figure 4.3 compares the Dropbox and YouTube share of the total traffic volume in *Campus 2*. Apart from the variation reflecting the weekly and holiday pattern, a high fraction is seen for Dropbox daily. Note that in this network the traffic exchanged with Dropbox is close to 100 GB per working day: that is already 4 % of all traffic, or a volume equivalent to about one third of YouTube traffic in the same day!

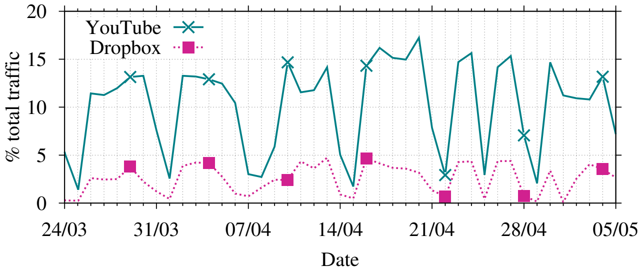


Figure 4.3: YouTube and Dropbox in *Campus 2* (in 2012).

These findings highlight an increasing interest for cloud storage services, showing that people are eager to make use of remote storage space. Cloud-based storage is already popular, with 6–12 % of home users regularly accessing one or more of the services. We restrict our attention to Dropbox only in the following, since it is by far the mostly used service in terms of traffic volume at the moment. Other providers will be evaluated in Chapter 5 by means of active experiments, while discussing the performance implications of design choices.

## 4.4 Dropbox Performance

### 4.4.1 Traffic Breakdown: Storage and Control

To understand the performance of the Dropbox service, we first study the amount of traffic handled by the different sets of servers. Figure 4.4 shows the resulting traffic breakdown in terms of traffic volume and number of flows. From the figure, it emerges that the Dropbox client application is responsible for more than 80 % of the total Dropbox traffic volume in all vantage points, which shows that this application is highly preferred over the Dropbox Web interfaces for exchanging data. A significant portion of the volume (from 7 % to 10 %) is generated by direct link downloads and the main Web interface (both represented as *Web* in Figure 4.4). In home networks, a small but non-negligible volume (up to 4 %) is seen to the Dropbox API, which is used by mobile devices. Finally, the data volume caused by control messages is negligible in all datasets.

The breakdown of the number of flows confirms that the Dropbox native client is the mostly used interface of the service. Client control servers are the major contributors: more than 80 % of the flows, depending on the dataset. The difference on the percentage of notification flows – around 15 % in *Campus 2*, *Home 1* and *Home 2*, and less than 3 % in *Campus 1* – is caused by the differ-

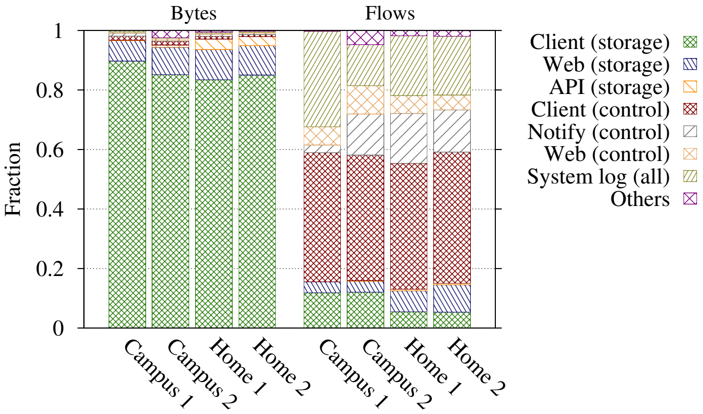


Figure 4.4: Traffic share of Dropbox servers.

ence in the typical duration of Dropbox sessions in those networks, which will be further studied in Section 4.5.4. Overall, the higher number of control and notification flows when compared to the storage ones exemplifies the signaling costs of using different servers for each specific functionality (see Table 4.1).

## 4.4.2 Server Locations and RTT

We showed in previous sections that the Dropbox client protocol relies on different servers to accomplish typical tasks such as file synchronization. Dropbox distributes the load among its servers both by rotating IP addresses in DNS responses and by providing different lists of DNS names to each client. In the following, we want to understand the geographical deployment of this architecture and its consequences on the perceived RTT.

### Server Locations

Names in Table 4.1 terminating in a numerical suffix are normally resolved to a single server IP address[4] and clients are in charge of selecting which server will be used in a request. For instance, meta-data servers are currently addressed by a fixed pool of 10 IP addresses and notification servers by a pool of 20 IP addresses. Storage servers are addressed by more than 600 IP addresses from Amazon data centers. Figure 4.5 shows the number of contacted storage servers per day in our vantage points. The figure points out that clients in *Campus 1*

[4] Meta-data servers are addressed in both ways, depending on the executed command, via `client-lb` or `clientX`.

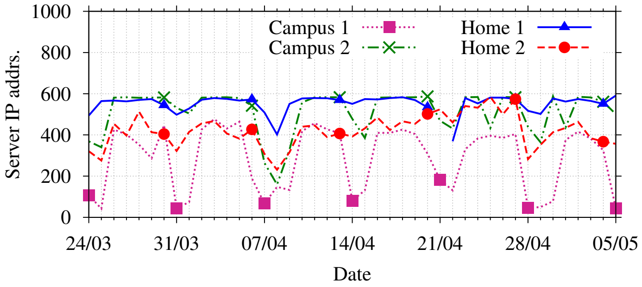


Figure 4.5: Number of contacted storage servers.

and *Home 2* do not reach all storage servers daily. In both *Campus 2* and *Home 1*, more servers are instead contacted because of the higher number of devices on those vantage points. Routing information shows that the control servers contacted from the monitored networks are all located in the U.S. West Coast (likely in California), while storage servers are in the U.S. East Coast (in Amazon's Northern Virginia data centers).

### Storage and Control RTT

A deeper analysis of the RTT at our four vantage points reveals more details of the physical implementation of the Dropbox architecture. Section 4.4.4 will show that the RTT has a major impact on the service performance. Figure 4.6 shows, separately for storage (left) and control flows (right), the CDF of the minimum RTT in flows where at least 10 RTT samples could be obtained (see [107]). The figure accounts only for the RTT between our probes and the servers, to filter out the impact of the access technologies (*e.g.,* ADSL).

The RTTs to storage servers at Amazon remained stable during our measurements, meaning that no significant changes in the network topology happened. The differences in RTTs among the vantage points are related to the countries where the probes are located. This constant RTT during our 42 days of measurements is another strong indication that a single data center is used by all users in our vantage points.

The RTTs to the control servers are more variable. In both *Campus 1* and *Home 2*, the curve presents small steps (less than 10 ms). We assume that they are caused by changes in the IP route, since the same behavior is not noticeable in all probes. Finally, it is interesting to note the high difference in the RTT between control and storage data centers, which is likely caused by the physical distance between them inside the U.S.

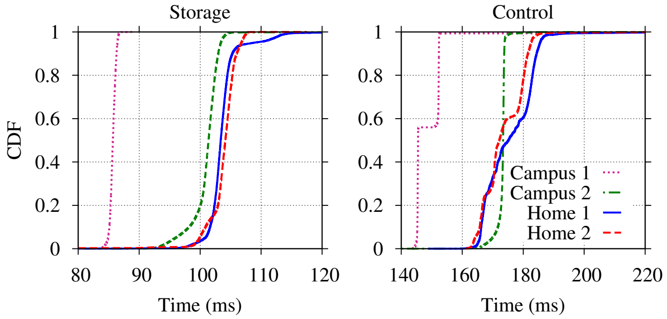


Figure 4.6: Minimum RTT of storage and control flows. Note the $x$-axes.

These results suggest that Dropbox is, as for now, a service centralized in the U.S. Considering that more than half of the Dropbox clients are outside the U.S.[48], and the high amount of traffic observed in our vantage points, the traffic exchanged between the clients and the data centers is likely to be already very relevant in the core network. In order to verify the Dropbox setup worldwide and compare it to alternative designs, Chapter 5 will perform active measurements using the PlanetLab. The remainder of this section concentrates on understanding the implications of Dropbox architecture for users in our datasets.

### 4.4.3 *Store* and *Retrieve* Flows

#### Flow Size

As shown in Figure 4.4, most Dropbox traffic is generated by file storage operations. Figure 4.7 depicts the CDF of the flow size for storage operations. Since TLS/SSL is used, we observe a minimum flow size of approximately 4 kB. The flows have a maximum size of approximately 400 MB because of current runtime parameters of Dropbox: batches are limited to a maximum of 100 chunks, each smaller than 4 MB, as described in Section 4.1.

From Figure 4.7 it emerges that a significant percentage of flows (up to 40 % in some cases) exchange less than 10 kB, meaning that they are composed mostly of TLS/SSL handshake messages and a small amount of user data. A very high percentage of flows (varying from 40 % to 80 %) consist of less than 100 kB. We conjecture that two factors are causing this behavior: (i) the synchronization protocol sending and receiving file *deltas* as soon as they are detected; (ii) the primary use of Dropbox for synchronization of small files, instead of periodic (large) backups. Chapter 5 will investigate this conjecture further by means of an experiment with volunteering Dropbox users.

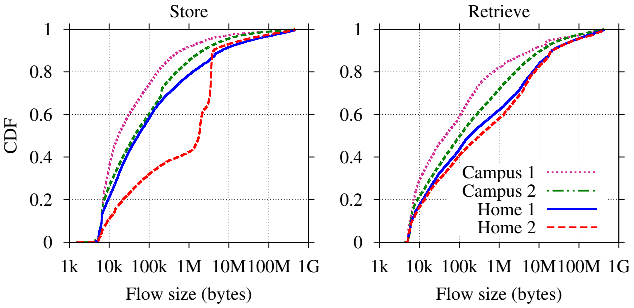


Figure 4.7: Distribution of TCP flow sizes of file storage for the Dropbox client.

Comparing the CDFs for the *store* and *retrieve* operations, we can see that *retrieve* flows are normally larger than the *store* ones. This is particularly visible in *Campus 1*, *Campus 2* and *Home 1* datasets. For instance, while 60 % of *store* flows in *Home 1* have no more than 100 kB, the percentage is about 40 % for *retrieve* flows in the same network. This can be partially explained by the first batch synchronization happening when sessions are started. Besides that, we observe a high number of devices using Dropbox only for downloading content. This usage will be analyzed further in the coming sections.

We also highlight a remarkably discrepancy in the CDF for *store* flows in *Home 2*. A single device submitted chunks in consecutive TCP connections during several days in our capture. This causes the CDF to be biased toward the maximum chunk size used by Dropbox (4 MB). The traffic characteristics suggest that chunks submitted by this device have never been acknowledged by Dropbox servers, pointing to an isolated problem manifested in this single device. This example illustrates the level of details that can be achieved using solely flow measurements to monitor cloud services, provided that application protocols are known and the network is properly instrumented.

### Chunks per Batch

Figure 4.8 depicts the CDF of the estimated number of chunks per flow. The curves show that most batches are composed of a small number of chunks. Storage flows have no more than 10 chunks in more than 80 % of the cases in all datasets. The *Home 2* distribution diverges because of the single client behaving abnormally described in the previous section. These distributions reinforce our conjecture about the dominance of deltas and small files in Dropbox usage habits: most flows are very small and composed of few chunks. Most of the remaining flows have the maximum allowed number of chunks per batch

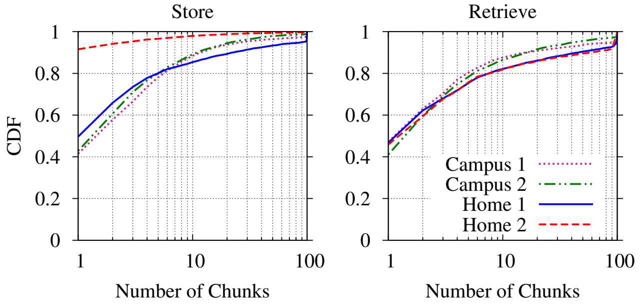


Figure 4.8: Distribution of the estimated number of file chunks per TCP flow.

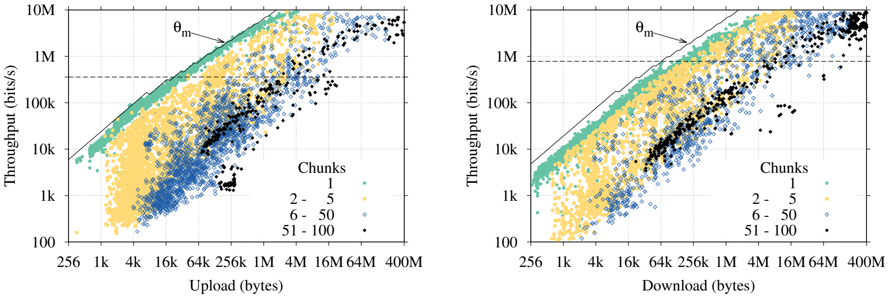
and, as such, are strongly shaped by the protocol design of Dropbox. Cloud storage developers should, therefore, design protocols that are efficient to handle workloads with these characteristics. Next, we evaluate the responsiveness [146] of Dropbox, represented by the system throughput.

### 4.4.4 Storage Throughput

Our measurements in Section 4.4.2 indicate that Dropbox relies on centralized data centers for control and storage. This raises the question on the service performance for users located far from those data centers. The throughput of storage operations is certainly a good metric to indicate to the responsiveness of cloud storage services, since it is related to both the latency (*i.e.,* it is also affected by the locality) and the size of storage operations.

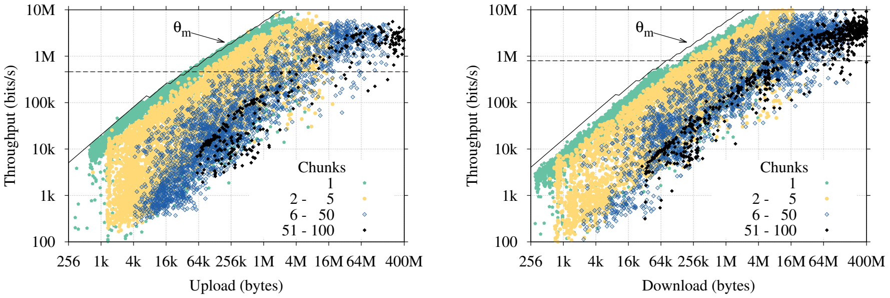
Figures 4.9 and 4.10 depict the throughput achieved by each storage flow in *Campus 1* and *Campus 2*, respectively. The figures show separate plots for the *store* and *retrieve* operations. *Home 1* and *Home 2* are left out of this analysis since the access technology (ADSL, in particular) is a bottleneck for the service in those networks. The $x$-axis represents the number of bytes $S$ transferred in the flow, already subtracting the typical TLS/SSL overheads, while the $y$-axis shows the throughput $\theta$ calculated as the ratio between transferred bytes $S$ and duration $\Delta t$ of each flow – *i.e.,* $\theta = S/\Delta t$ (note the logarithmic scales). The duration $\Delta t$ is accounted as the time between the first TCP `SYN` packet and the last packet with payload in the flow, ignoring connection termination delays. Flows are represented by different marks in the figure according to their number of chunks. We refer to Appendix B for more details on our methodology to calculate the storage throughput.

Overall, the throughput is remarkably low. The average throughput (marked with dashed horizontal lines in the figures) is not higher than 462 kbits/s for *store*



(a) Store (b) Retrieve

Figure 4.9: Throughput of storage flows in *Campus 1*.



(a) Store (b) Retrieve

Figure 4.10: Throughput of storage flows in *Campus 2*.

flows and 797 kbits/s for *retrieve* flows in *Campus 2* (359 kbits/s and 783 kbits/s in *Campus 1*, respectively). In general, the highest observed throughput (close to 10 Mbits/s in both datasets) is only achieved by flows carrying more than 1 MB. Moreover, flows achieve lower throughput as the number of chunks increases. This can be seen by the concentration of flows with high number of chunks in the bottom part of the plots for any given size.

TCP start-up times and application layer sequential acknowledgments are two major factors limiting the throughput, affecting flows with a small amount of data and flows with a large number of chunks, respectively. In both cases, the high RTT between clients and data centers amplifies the effects. In the following, those effects are detailed.

### TCP Start-up Effects

Flows carrying a small amount of data are limited by TCP slow start-up mechanism. This is particularly relevant in the analyzed campus networks, since both data link capacity and RTT to storage data centers are high in these networks. Figures 4.9 and 4.10 show the maximum throughput $\theta_m = S/L$ for completing the transfer of a specific amount of data $S$, assuming that flows stay in the TCP slow start phase. We compute the latency $L$ for completing the transfer using the following formula [53]:

$$L = \left\lceil \log_\gamma \left( \frac{S(\gamma - 1)}{init\_cwnd} + 1 \right) \right\rceil \times RTT + \frac{S}{C} \tag{4.1}$$

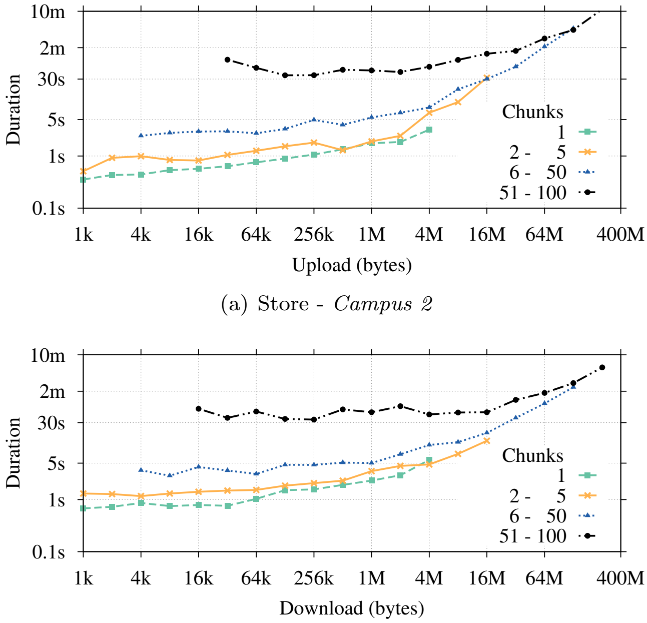
where $S$ in the formula is again the flow size, $C$ is the bottleneck network link-rate and $\gamma = 1.5$, since delayed acknowledgments are assumed. Moreover, the initial congestion window (*init_cwnd*) is set to 3, and the total latency of a flow is adjusted to include constant overheads – *e.g.,* the 3 RTTs of TLS/SSL handshakes in the Dropbox setup observed during our data capture (see Appendix B).

Figures 4.9 and 4.10 show that $\theta$ approximates the maximum throughput satisfactorily. This is clearer for flows with a single chunk, which suffer less from application layer impediments. The bound is less tight for *retrieve* flows, because their latency includes at least 1 server reaction time. Note that the throughput can still be limited by other factors, such as the explicit user selection of transfer limits or possible network congestion. However, when considering only flows with a single chunk, more than 99 % of the *store* flows and around 95 % of the *retrieve* flows in *Campus 1* have no TCP retransmissions. These percentages are lower in *Campus 2* (88 % and 75 %) because of the wireless access points. Single-chunk flows with no retransmissions experience throughput close to the theoretical limit $\theta_m$[5] confirming that the TCP slow-start is their main bottleneck.

### Sequential Acknowledgments

The use of sequential acknowledgments at the application layer by Dropbox 1.2.52 (see Figure4.1) is a major bottleneck for flows with more than 1 chunk. The mechanism forces clients to wait one RTT (plus the server reaction time) between two storage operations. Naturally, flows carrying a large number of small chunks suffer relatively more from this impediment than flows with large chunks. Figure 4.8 shows that more than 40 % of the flows have at least 2 chunks and are potentially affected.

[5] The average throughput of *store* flows in *Campus 1* is only 17 % lower than what would be obtained if each flow would have experienced the maximum theoretical throughput.



(a) Store - *Campus 2*

(b) Retrieve - *Campus 2*

Figure 4.11: Minimum duration of flows with diverse number of chunks.

Flows with several chunks are also affected by the previously described factors. Besides that, the Dropbox client keeps storage connections open for a short interval after transferring a chunk, waiting for new chunks. Therefore, some flows in Figures 4.9 and 4.10 might have a longer duration because they are reused during this idle interval. To remove these effects and highlight the impact of sequential acknowledgments, we divide the $x$-axis of Figure 4.10 in slots of equal sizes (in logarithmic scale) and, for each slot, select the flow with maximum throughput in each of the four groups shown in the figure. Figure 4.11 depicts the duration of these representative flows. Flows with more than 50 chunks, for instance, always last for more than 30s, regardless of their size. Considering the RTT in *Campus 2* (around 100 ms), up to one third of these 30 s (5-10 s, since 1 RTT is waited between chunks) is wasted while application layer acknowledgments are transiting the network. The remaining duration is mainly caused by the server and the client reaction times between chunks.

### 4.4.5 Implications

Our measurements clearly indicate that the application layer protocol in combination with large RTT penalizes the service performance. We identify three possible solutions to remove the identified bottlenecks:

1. Bundling smaller chunks, thus increasing the amount of data sent per storage operation. Dropbox 1.4.0, announced in April 2012, implements a bundling mechanism, which is analyzed in the following;

2. Using a delayed acknowledgment scheme in storage operations, thus pipelining chunks to remove the effects of sequential acknowledgments;

3. Bringing storage servers closer to customers, thus reducing the latency and, by consequence, improving the overall throughput.

Note that the first two countermeasures target only the application layer bottleneck. The third one, while valid for any on-line service, would have the extra positive impact of removing storage traffic from the core of the Internet.

#### Improvements in Dropbox 1.4.0

Dropbox 1.4.0 adds new protocol commands, allowing several chunks to be submitted in a single application layer operation. We use the extra data captured in *Campus 1* during Jun and Jul 2012 to quantify the effects of this mechanism on the service performance.

Table 4.4 compares flow size and throughput distributions before and after the deployment of Dropbox 1.4.0. The increase in the median flow size shows that flows are bigger in this version, because small chunks can be accommodated in a single TCP connection. The averages are less affected, since only small flows profit from the mechanism. Both the median and the average throughput, on the other hand, are dramatically improved. The average throughput of *retrieve* flows, for instance, is around 65 % higher in the newest dataset!

Since both the new batch commands and the old single-chunk commands are still executed sequentially, there seems to exist further room for improvement in the protocol. Overall, these results show the importance of protocol design to the performance of cloud storage. The lessons learned are important for engineers building new cloud storage services. The next chapter will complement the analysis of storage protocols by comparing the design of Dropbox to that of its competitors.

Table 4.4: *Campus 1* before and after the deployment of a bundling mechanism.

| | Mar/Apr | | Jun/Jul | |
|---|---|---|---|---|
| | Median | Average | Median | Average |
| Flow size | | | | |
| *Store* | 16.28 kB | 3.91 MB | 42.36 kB | 4.35 MB |
| *Retrieve* | 42.20 kB | 8.57 MB | 70.69 kB | 9.36 MB |
| Throughput (kbits/s) | | | | |
| *Store* | 31.59 | 358.17 | 81.82 | 552.92 |
| *Retrieve* | 57.72 | 782.99 | 109.92 | 1293.72 |

## 4.5 Service Usage and Workload

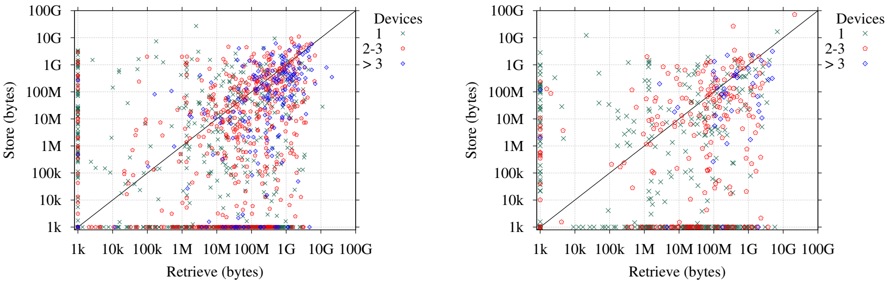
After studying the Dropbox architecture and revealing several performance bottlenecks from its protocol design, we turn our attention to the service usage. This section describes several characteristics about Dropbox traffic, aiming to provide a reference on typical workloads, which can help to model and dimension the infrastructure for similar services.

Section 4.5.1 studies Dropbox users' behavior, revealing typical usage scenarios and their consequences in terms of network traffic. After that, Section 4.5.2 and Section 4.5.3 study the distributions of devices per household and shared folders per user. Section 4.5.4 and Section 4.5.5 characterize Dropbox session duration and usage seasonality, respectively. While all these sections focus on the Dropbox client software, Section 4.5.6 discusses the (less popular) Dropbox Web interface. Finally, Section 4.5.7 summarizes the findings in this section.

### 4.5.1 Storage Volume

In this section we correlate ISP customers (IP addresses) in home networks and the total storage volume in *store* and *retrieve* operations. Both campuses are left out of this analysis because IP addresses are not correlated to users or households in those networks. The amount of stored and retrieved data per household is depicted in Figure 4.12. Each IP address is represented by a point and different symbols are used to illustrate the number of devices behind the IP address. Note that we place all cases with less than 1 kB on the axes because of the logarithmic scales. The figure accounts only for transfers made from the Dropbox client. Similarly to Section 4.4.4, the typical overhead of TLS/SSL negotiations are subtracted from the transferred amount.

Figure 4.12 shows that Dropbox users tend to download more than upload. This is visible in the higher density of points below the diagonals. Overall, the total downloaded data in *Home 1* is 1.4 times higher than the total uploaded data



(a) *Home 1*

(b) *Home 2*

Figure 4.12: Data volume stored and retrieved from Dropbox.

(for illustration, this ratio is 1.6 in *Campus 1* and 2.4 in *Campus 2*). *Home 2* is an exception: the ratio is around 0.9 in this network. Some customers massively uploading content create this divergence. These customers appear on the top right corner of Figure 4.12(b). Note that one of these customers is also responsible for the bias in the CDF depicted in Figure 4.7.

Four usage scenarios can be identified: (i) *occasional* users, which abandon their Dropbox clients, hardly synchronizing any content (points close to the origin); (ii) *upload-only* users that mainly submit files (points close to the $y$-axes); (iii) *download-only* users, executing predominantly *retrieve* operations (points close to the $x$-axes); (iv) *heavy* users that both store and retrieve large amounts of data (points along the diagonals). The proportion of users in each group explains the overall relation between downloads and uploads.

Table 4.5 quantifies the groups. The IP addresses are divided according to the following heuristics: IP addresses that have less than 10 kB in both *retrieve* and *store* operations are included in the *occasional* group; IP addresses that have more than three orders of magnitude of difference between upload and download (*e.g.,* 1 GB versus 1 MB) are included in either *download-only* or *upload-only*; all others are in the *heavy* group. For each group, the table shows the percentage of IP addresses and sessions, the total transferred data, the average number of days devices are seen on-line, and the average number of devices per household.

The *occasional* group represents around 30 % of the total IP addresses in both vantage points. As expected, customers in this group exchange a negligible amount of data and stay on-line in Dropbox less time when compared to others. They also have the lowest average number of devices. This group, therefore, marginally generates any load to the system.

The *upload-only* group accounts for around 7 % of the IP addresses, and is responsible for a significant amount of uploads (21 % in *Home 1* and 11 % in

Table 4.5: Fraction of IP addresses and sessions, retrieved and stored data volume, average number of days on-line, and average number of devices of the different user groups in *Home 1* and *Home 2*.

(a) Home 1

| Group | Addresses | Sessions | Retrieve | Store | Days | Devices |
|---|---|---|---|---|---|---|
| Occasional | 31 % | 15 % | - | - | 16.37 | 1.22 |
| Upload-only | 6 % | 6 % | - | 84 GB | 19.74 | 1.36 |
| Download-only | 26 % | 24 % | 135 GB | - | 21.53 | 1.69 |
| Heavy | 37 % | 54 % | 417 GB | 321 GB | 27.54 | 2.65 |

(b) Home 2

| Group | Addresses | Sessions | Retrieve | Store | Days | Devices |
|---|---|---|---|---|---|---|
| Occasional | 32 % | 18 % | - | - | 15.52 | 1.13 |
| Upload-only | 7 % | 4 % | - | 26 GB | 20.42 | 1.37 |
| Download-only | 28 % | 23 % | 57 GB | - | 17.37 | 1.34 |
| Heavy | 33 % | 55 % | 147 GB | 193 GB | 27.10 | 2.16 |

*Home 2*). Considering their low number of devices, users in this group seem to be interested in Dropbox for backups and for the submission of content to third-parties or to geographically dispersed devices. The opposite behavior can be concluded for the *download-only* group – *e.g.,* people that use Dropbox to receive content produced in other locations. This group is, however, very significant in both number of IP addresses (26 % in *Home 1* and 28 % in *Home 2*) and transferred volume (25 % and 28 %, respectively). Similarly to the *upload-only* group, a moderate number of devices per IP address is seen in this group.

Finally, accounting for 37 % of IP addresses in *Home 1* and 33 % in *Home 2*, the *heavy* group is responsible for most of the volume transferred by Dropbox clients. Customers in this group have a high number of devices (above 2 on average), appear on-line more than 60 % of the days in our capture and are responsible for more than 50 % of the Dropbox sessions. The usage of Dropbox for synchronization of devices in a household seems to be the typical scenario in this group. These are, therefore, the users causing the biggest impact on both system workload and network utilization.

### 4.5.2 Devices

Devices connected to the same LAN can use the LAN Sync Protocol for synchronizing files without retrieving duplicated content from the cloud, thus saving both network and server resources. We describe the distribution of the number

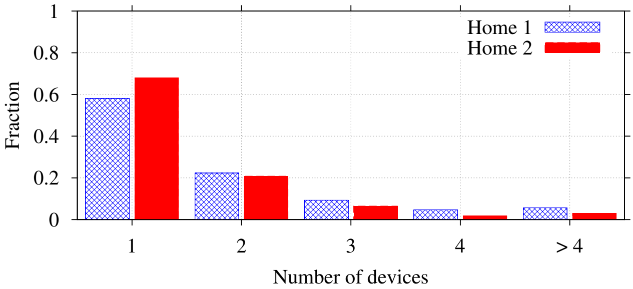


Figure 4.13: Distribution of the number of Dropbox devices per household.

of devices residing in the same LAN, aiming to check the potential for savings by implementing LAN Sync capabilities. Figure 4.13 depicts the distribution of the number of devices per IP address in *Home 1* and *Home 2*.

In around 60 % of the households using the Dropbox client, there is only a single device linked to the service. Most of the remaining households have up to 4 devices and, not surprisingly, are part of the *heavy* group identified in the previous section. By inspecting a subset of notification connections in *Home 1*, we observe that in around 60 % of households with more than 1 device (around 25 % of the total), at least 1 folder is shared among the devices. This further confirms our findings about the typical use of Dropbox among *heavy* users for the synchronization of devices. Since the LAN Sync Protocol traffic does not reach our probes, we cannot precisely quantify the amount of bandwidth saved in these households by the use of the protocol. We can conclude, however, that relatively few opportunities for using the LAN Sync Protocol appear in practice, with no more than 25 % of the households profiting from that. The remaining users always rely on central storage data centers for their data transfers.

### 4.5.3 Shared Folders

Next, we measure to what extent Dropbox is being used for content sharing or collaborative work. Different users sharing folders can profit from the LAN Sync Protocol as well. Besides that, providers can deploy special capabilities to save storage space, if several users share the same content. We analyze namespace identifications (*i.e.*, folder identifiers) in *Home 1* and *Campus 1* traffic.[6] Figure 4.14 shows the distribution of the number of namespaces per device. By analyzing *Campus 1* data on different dates, it is possible to conclude that the

[6] This information could not exposed in *Home 2* and *Campus 2* for technical reasons.

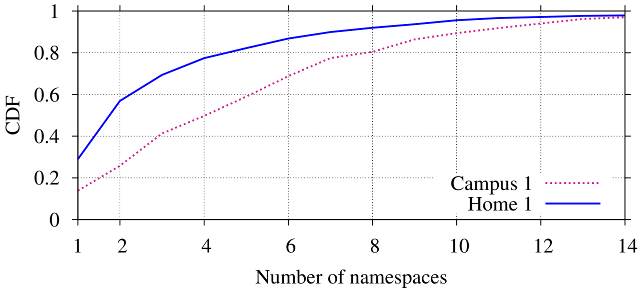


Figure 4.14: Number of namespaces per device.

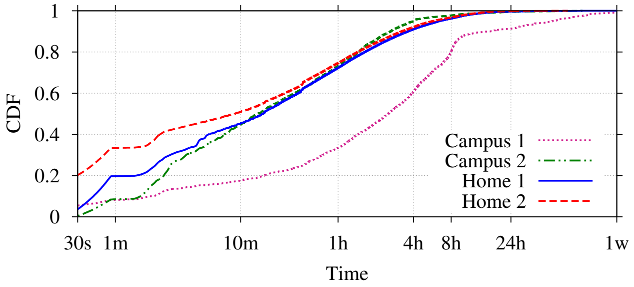


Figure 4.15: Distribution of session durations.

number of namespaces per device is not stationary and has a slightly increasing trend. Figure 4.14 is built considering the last observed number of namespaces on each device in our datasets.

In both networks the number of users with only 1 namespace (the users' root folder) is small: 13 % in *Campus 1* and 28 % in *Home 1*. In general, users in *Campus 1* have more namespaces than in *Home 1*. The percentage of users having 5 or more namespaces is equal to 50 % in the former, and 23 % in the latter. When considering only IP addresses assigned to workstations in *Campus 1*, each device has on average 3.86 namespaces. These results suggest that content sharing is common, and providers should prepare their solutions to handle content duplication. Chapter 5 will discuss such aspects further, using data from volunteering Dropbox users.

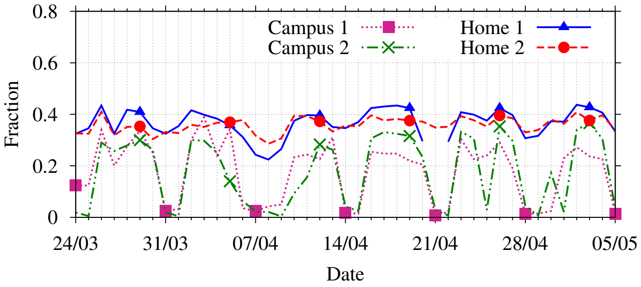


Figure 4.16: Device start-ups: fraction of the number of devices in the probe.

### 4.5.4 Session Duration

We analyze the session duration based on the TCP flows to notification servers. *Home 1*, *Home 2*, and *Campus 2* have a similar behavior in general, as shown in Figure 4.15, with an exception for the number of short-lived sessions. In both home networks, a significant number of notification flows are terminated in less than 1 minute. A closer inspection reveals that most of those flows are from few devices. Their divergent TCP behavior suggests that network equipment (*e.g.* NAT or firewalls – see [83]) might be terminating notification connections abruptly. Considering the normal operation of the Dropbox protocol, notification connections are re-established immediately after that.

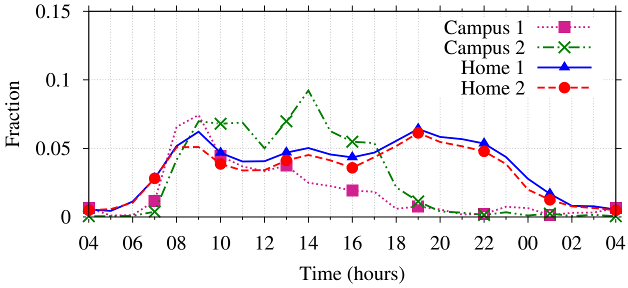
Most devices in *Home 1*, *Home 2* and *Campus 2* stay connected up to 4 hours in a single session. In *Campus 1*, a significantly higher percentage of long-lasting sessions is seen. This can be explained by the prevalence of workstations in a typical 8-hours work routine. Inflections at the tail of the distributions are seen in all curves, as a consequence of the devices kept always on-line.

### 4.5.5 Daily Usage

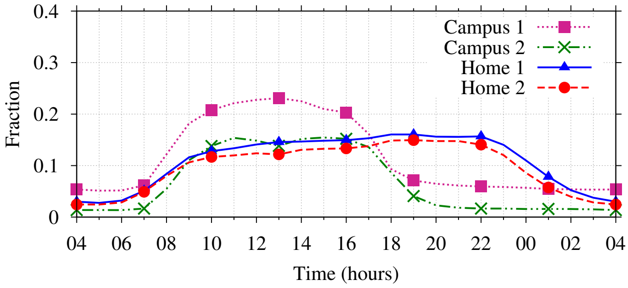
We characterize whether the use of the Dropbox client has any typical seasonality. Figure 4.16 shows the time series of the number of distinct devices starting up a Dropbox session in each vantage point per day. The time series are normalized by the total number of devices in each dataset. Around 40 % of all devices start at least one session every day in home networks, including weekends.[7] In campus networks, on the other hand, there is a strong weekly seasonality.

At a finer time scale (1 hour bins), we observe that the service usage follows a clear day-night pattern. Figure 4.17 depicts the daily usage of the Dropbox

[7] Note the exceptions around holidays in April and May.



(a) Session start-ups



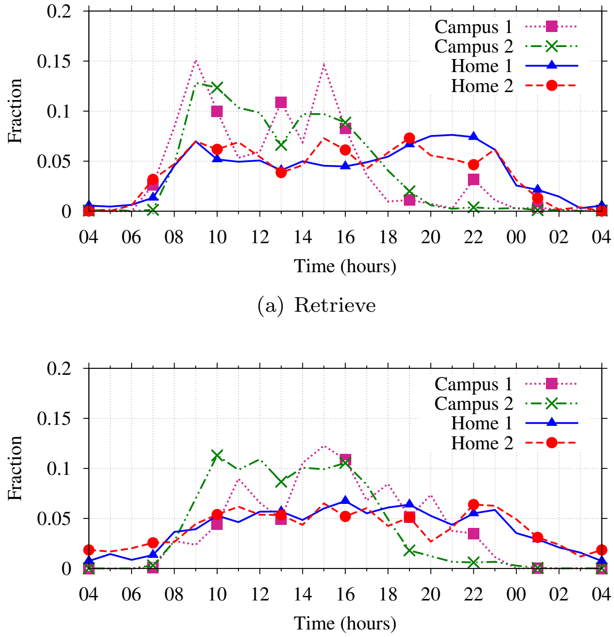
(b) Active devices

Figure 4.17: Daily start-ups and sessions on weekdays.

client. All plots are produced by averaging the quantities per interval across all working days in our datasets.

Figure 4.17(a) shows the fraction of distinct devices that start a session in each interval, while Figure 4.17(b) depicts the fraction of devices that are active (*i.e.,* are connected to Dropbox) per time interval. From these figures we can see that Dropbox usage varies strongly in different locations, following the presence of users in the environment. For instance, in *Campus 1*, session start-ups have a clear relation with employees' office hours. Session start-ups are better distributed during the day in *Campus 2* as a consequence of the transit of students at wireless access points. In home networks, peaks of start-ups are seen early in the morning and during the evenings. Overall, all time series of active devices (Figure 4.17(b)) are smooth, showing that the number of active users at any time of the day is easily predictable.

Figure 4.18 depicts the fraction of the total number of bytes exchanged in each time interval in *retrieve* and *store* functions. Figure 4.18(a) shows that



(a) Retrieve

(b) Store

Figure 4.18: Daily usage of Dropbox storage on weekdays.

the number of bytes received in *retrieve* operations has a correlation with client start-ups. This suggests that the first synchronization after starting a device is dominated by the download of content produced elsewhere, instead of the upload of content produced off-line. Although other patterns are visible in the figures, such as the concentration of downloads in the morning in *Campus 1* and in the evening in *Home 1*, both series are still noisy at this level of aggregation.

### 4.5.6 Web Storage

In addition to the client application, Dropbox allows users to access shared folders and files using both its main Web interface and a direct link download mechanism. In the following, we analyze the usage of these interfaces. Figure 4.19 presents the CDFs of the number of bytes in the storage flows of the Dropbox main Web interface. Separate CDFs for uploads and downloads are presented.

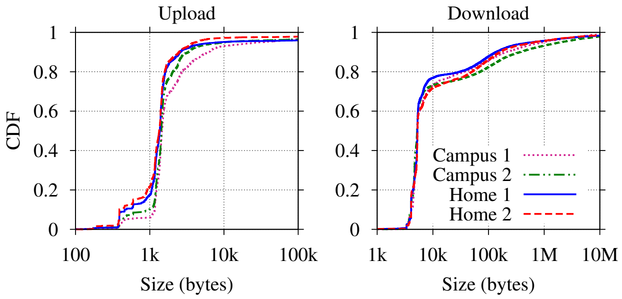


Figure 4.19: Storage via the main Web interface.

Considering the number of uploaded bytes, it becomes clear that the main Web interface is hardly used for uploading content. More than 95 % of the flows submitted less than 10 kB. When considering downloaded bytes, up to 80 % of flows exchanged less 10 kB. These distributions are, however, strongly biased toward the TLS/SSL handshake sizes for two reasons: (i) the Dropbox interface retrieves thumbnails from storage servers using TLS/SSL; (ii) Web browsers open several parallel connections when retrieving those HTTP objects. The remaining flows have less than 10 MB in more than 95 % of the cases, showing that only small files are normally retrieved from this Web interface.

Additionally, we analyze flows related to direct link downloads. Note that these flows correspond to 92 % of the Dropbox Web storage flows in *Home 1*, confirming that this mechanism is highly preferred over the main Dropbox Web interface. Figure 4.20 shows the CDF of the size of direct link downloads.[8] Since these downloads are not always encrypted, the CDF does not have the TLS/SSL lower-bound. Interestingly, only a small percentage of direct link downloads is bigger than 10 MB, suggesting that their usage is not related to the sharing of movies or large archives.

### 4.5.7 Implications

This section helps to understand the usage of Dropbox. This is needed, for instance, for provisioning data centers and networks to handle cloud storage traffic. Our analysis of user behaviors reveals that users have different interests regarding the application. For instance, although Dropbox is already installed in more than 6 % of the households (see Section 4.3), less than 40 % of these households are fully using its functionalities – *i.e.*, synchronizing devices, sharing folders

[8] Uploads are not shown since a single HTTP request is sent. *Campus 2* is not depicted due to the lack of FQDN.

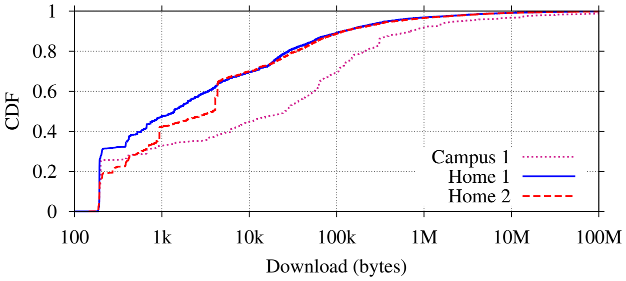


Figure 4.20: Size of direct link downloads.

etc. Interestingly, the results are similar in both home networks, reinforcing our conclusions and showing that cloud storage traffic is very predictable. The high amount of traffic created by this small percentage of users motivates our expectation that cloud storage services will be among the top applications producing Internet traffic soon. Geographically dispersed sources as well as longitudinal data are, however, necessary to check whether the conclusions of this section can be generalized, when more people adopt such solutions.

## 4.6 Conclusions

The first contribution of this chapter is the extension of the general methodology developed in Chapter 3 to monitor performance of cloud services. This was achieved by introducing a method for both (i) isolating cloud storage traffic; and (ii) calculating performance metrics specific to Dropbox. Both steps were possible because we first succeeded in reverse-engineering the Dropbox protocols. Then, we relied on a specialized flow exporter to collect the measurements required to our analysis. Results in this chapter showed that flow measurements are suitable for monitoring advanced aspects of cloud services, provided that the measurement environment is carefully prepared and the network is properly instrumented. However, limitations of the approach also became evident, such as the need for detailed knowledge of proprietary protocols, which is by no means trivial to be obtained.

The application of our method resulted in the first study on the usage of Dropbox on the Internet. By analyzing flows captured at 4 vantage points in Europe over a period of 42 days, we performed an extensive characterization of the service, both in terms of the system workload and the typical usage. The main contributions of this analysis are:

- We quantified the usage of cloud storage in operational networks. Our measurements showed a significant amount of traffic related to the services, especially on campus networks. We expect these services to become popular at home too, where penetration is already above 6 %.

- We highlighted that Dropbox performance is mainly driven by the distance between clients and storage data centers. In addition, short data transfer sizes coupled with a per-chunk acknowledgment mechanism impair transfer throughput.

- In terms of workloads, our analysis showed how user behavior is reflected in network traffic. For instance, a considerable number of users takes full advantage of the Dropbox functionalities, actively storing files and sharing several folders. However, we also noted around one third of users completely abandoning their clients, seldom exchanging any data during the 42 days of observations.

Regarding the protocol bottlenecks, we identified two possible improvements: (i) the use of a file bundling scheme; (ii) the introduction of delayed acknowledgments. We showed that the recent deployment of a bundling mechanism already improves Dropbox performance dramatically. In addition, we expect that the overall performance will be improved by the deployment of other data centers in different locations.

Finally, this chapter also contributed with the first large-scale public datasets reporting the activity of *anonymized* Dropbox users. Our datasets and scripts can be downloaded from the SimpleWeb trace repository at `http://www.simpleweb.org/wiki/Dropbox_Traces`

CHAPTER 5

# Comparing Cloud Storage Services

The cloud-based architecture in conjunction with the proprietary protocols of cloud storage services result in a black-box approach toward cloud storage. Results in Chapter 4 about Dropbox indicated that design and architectural choices strongly influence cloud storage performance and network usage. However, very little is known about how different providers (*i.e.,* other than Dropbox) implement their services and – most of all – performance implications of design choices. This understanding is valuable as a guideline for building well-performing services that wisely use network resources.

The goal of this chapter is twofold. Firstly, we investigate how different providers tackle the problem of synchronizing files. For answering this question, we develop a methodology that helps to understand both system architecture and client capabilities. We apply our methodology to compare five services (see Table 5.1), revealing differences on client software, synchronization protocols and data center placement. Secondly, we investigate the consequences of such designs on performance. Taking the perspective of users connected from a single location in Europe, we benchmark each selected service under the same conditions, highlighting bottlenecks in various usage scenarios and emphasizing the relevance of design choices for both users and the Internet.

This chapter compares several cloud storage services using *active measurements.* Previous works have already studied cloud storage services [104, 148], but without investigating how different designs affect cloud storage performance. The authors of [101] benchmark cloud providers, but focusing only on server infrastructure. Similarly to our goal, [87] evaluates Dropbox, Mozy, Carbonite and CrashPlan. In contrast to the previous work and motivated by the extensive list of providers on the market, we first propose a methodology to automate the benchmarking. Then, we analyze several synchronization scenarios and providers, thus shedding light on the impact of design choices on performance.

Our results reveal interesting insights, such as unexpected drops in performance in common scenarios because of both the lack of client capabilities and architectural differences in the services. Overall, the lessons learned are useful as guidelines to the design of cloud storage services.

This chapter is further organized as follows. Section 5.1 describes our proposed methodology. Section 5.2 applies our methodology to unveil the protocols and data center locations of the analyzed services. Section 5.3 presents the outcome of a preliminary data collection with volunteers (*i.e.,* a crowd-sourcing), which is used to guide the definition of our benchmarks. Client capabilities are studied in Section 5.4, while Section 5.5 shows our benchmarking results. Finally, Section 5.6 concludes the chapter and summarizes our contributions.

## 5.1 Methodology

This section describes our methodology to study cloud storage services. The methodology is built around a testbed that is described in Section 5.1.1. The testbed is used to perform active measurements aiming at (i) reverse engineering protocols; (ii) checking data center locations; and (iii) benchmarking performance of storage services. Sections 5.1.2, 5.1.3 and 5.1.4 describe each of these parts of the methodology, respectively. Finally, Section 5.1.5 lists the services analyzed in the remainder of the chapter.

### 5.1.1 Testbed

Figure 5.1 depicts our testbed. It is composed of two parts: (i) the *test computers* that run the application-under-test in the desired operating system; and (ii) our *testing application.* The complete environment can run either in a single machine, or in separate machines provided that the testing application can intercept traffic from the test computers. In analogy to the setup used in previous chapters (see Figure 3.1 in Chapter 3), the testbed can also operate in cooperation with a specialized flow exporter, which takes over the tasks of intercepting packets and calculating statistics from the network traffic.

We build the testbed in a single Linux server for the experiments in the following. The Linux server both controls the experiments and hosts virtual machines that run the test computers (Windows 7 Enterprise).[1] Our testbed is connected to a 1 GB/s Ethernet network at the University of Twente, in which Internet connectivity is not a bottleneck.

Our testing application receives as input benchmarking parameters describing the sequence of operations to be performed (step 0 in Figure 5.1). The testing application acts remotely on a test computer, generating specific workloads in the form of file batches, which are manipulated using an FTP client (step 1). Files of different types are created or modified at run-time, *e.g.,* text

[1] OS X and Linux clients also have been checked if available and show no differences.

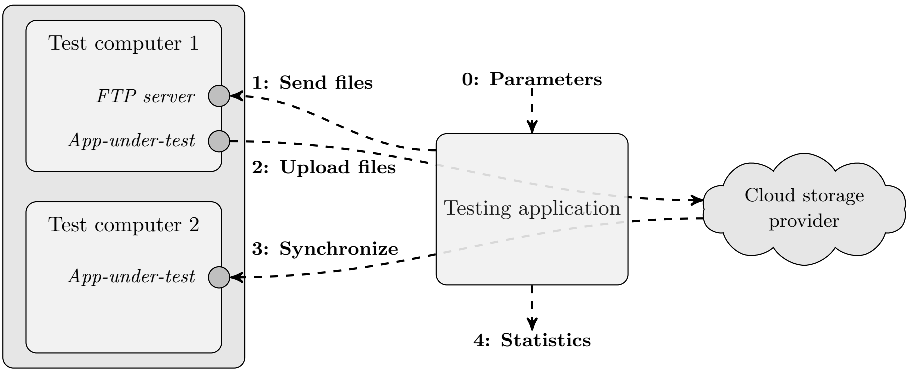


Figure 5.1: Testbed to study cloud storage services.

files composed of random words from a dictionary, images with random pixels, or random binary files. Generated files are uploaded to the cloud by the application-under-test (step 2). The application-under-test then synchronizes the files to other test computers in the testbed (step 3). The exchanged traffic is monitored during all actions to compute performance metrics (step 4). These metrics include the amount of traffic seen during the experiments and metrics related to the responsiveness of the services, such as the time before actual synchronization starts and the time to complete each synchronization step (*i.e.,* steps 2 and 3).

### 5.1.2 Unveiling Storage Protocols

Our testbed includes the methodology described in Chapter 4 to study cloud storage protocols, which relies on an *intercept proxy* – *i.e.,* a `Squid` proxy server extended with the module `SSL-bump` [124]. As in the case of Dropbox, other cloud storage clients need to be instructed to trust the self-signed proxy certificate. In contrast to Dropbox, in which trusted certificate authorities are hard-coded, other services can be trivially customized either by editing the operating system certificate stores, or by changing local (plain text) configuration files. By means of this setup, we are able to understand the client communication of the selected services and compare solutions of different providers to the same synchronization scenarios.

### 5.1.3 Architecture and Data Centers

The used architecture, data center locations and data center owner are important aspects of cloud storage services, having both legal and performance implications. To identify how the analyzed services operate, we observe the DNS name of contacted servers when (i) starting the application; (ii) immediately after files are manipulated; and (iii) when the application is in idle state. For each service, a list of contacted DNS names is compiled.

To reveal all IP addresses of the front-end nodes used by a service, DNS names are resolved to IP addresses by contacting more than 2,000 open DNS resolvers spread around the world.[2] In fact, cloud services rely on the DNS to distribute workload, returning different IP addresses according to the originating DNS resolver [14].

The owners of the IP addresses are identified using the `whois` service. For each IP address, we look for the geographic location of the server. Since popular geolocation databases are known to have serious limitations regarding cloud providers [118], we rely on a hybrid methodology that makes use of: (i) informative strings (*i.e.,* International Airport Codes) revealed by reverse DNS lookup; (ii) the shortest RTT to PlanetLab nodes [136]; and (iii) active `traceroute` to spot the closest well-known location of a router. Previous works [14, 55] indicate that these methods provide an estimation with about a hundred of kilometers of precision. Since this chapter aims at a coarse reference of the location from where cloud providers operate (*e.g.,* at country level), this estimation is sufficient for our goals.

### 5.1.4 Benchmarking Performance

Cloud storage applications can be implemented in different ways. We follow three steps in order to compare how different design choices impact performance.

Firstly, we carry a crowd-sourcing experiment to have a view on what type of files people store in the cloud. This has been done by means of a standalone application executed by volunteers. The application identifies automatically whether the volunteer uses Dropbox.[3] Then, all files in the volunteer's cloud storage folder are read, and file meta-data are extracted. The information is submitted to a centralized server, where it is anonymized and archived. More information about this experiment will be provided in Section 5.3.

Secondly, both Chapter 4 and our crowd-sourcing experiment show that cloud storage applications can optimize storage usage and speed up transfers

[2] The list has been manually compiled from various sources and covers more than 100 countries and 500 ISPs.

[3] We collect data from Dropbox users only for the sake of simplicity, under the assumption that usage is similar in other services.

Table 5.1: Analyzed cloud storage services.

| Name | Version |
|---|---|
| Dropbox | 2.0.8 |
| Microsoft SkyDrive | 1.8.4357.4863 |
| Google Drive | 17.0.2006.0314 |
| LaCie Wuala | *Strasbourg* |
| Amazon Cloud Drive | 2.0.2013.841 |

by implementing special synchronization capabilities. We develop experiments in our testbed to observe whether specific capabilities to enhance performance are present in the five selected services. We describe these experiments directly in Section 5.4, together with our results. In summary, our testing application produces batches of files that would benefit from each capability. The exchanged traffic is analyzed to determine how the services operate.

Finally, after knowing how the services are designed in terms of both data center locations and client capabilities, we quantify how such choices influence synchronization performance and the amount of overhead traffic. The results of our crowd-sourcing experiment are used to define benchmark sets, in which files of different sizes and formats are synchronized. Each tool is tested in several experiments and performance metrics are calculated. More details about these experiments will be provided in Section 5.5.

### 5.1.5 Tested Storage Services

We focus on five services for the sake of brevity, although our methodology is generic and can be applied to any other service. We restrict our analysis to native clients, since the previous chapter showed that this is the largely preferred means to use cloud storage services.

Table 5.1 lists the analyzed services. Dropbox [46], Google Drive [72] and SkyDrive [110] are selected because they are among the most popular offers, according to both results in Chapter 4 and the volume of search queries containing names of cloud storage services (see the Google Trends [74]). Wuala [95] is considered because it is a system that offers encryption at the client-side. We want to verify the impact of such a privacy/security enhancing technique on synchronization performance. Finally, we include Cloud Drive [3] to compare its performance to Dropbox, since both these services rely on AWS data centers.

## 5.2 System Architecture

This section provides a summary of the observed client protocols as well as data center locations. A complete description of client protocols is outside the scope of this thesis, since protocol details are likely to change over time. We, instead, focus on highlighting design elements that have significant implications in terms of performance.

### 5.2.1 Protocols

Two components can be identified in the five analyzed services (see Table 5.1): *control* and *data storage*. Regarding control, servers seem to perform three major tasks: *client authentication*, *file meta-data control*, and *notification* of changes to clients. All clients exchange traffic using HTTPS, with the exception of Dropbox notifications, which rely on plain HTTP. Interestingly, some Wuala storage operations also use HTTP, since users' privacy has already been secured by local encryption.

All services but Wuala use separate servers for control and storage. Except for Wuala, the identification of control and storage servers is trivial by monitoring the traffic exchanged when the clients (i) start; (ii) are idle; and (iii) synchronize files. Both server names (*i.e.,* FQDNs) and IP addresses differ while performing these operations and, therefore, can be used to isolate control and storage traffic. In the case of Wuala, we identify control and storage flows during our tests by relying on flow sizes and on the typical sequence that Wuala opens TCP connections.

We notice some relevant differences among the applications during login and idle phases. Figure 5.2 reports the cumulative number of bytes exchanged with control servers considering an initial 16 minutes after starting the clients and without synchronizing any files (*i.e.,* in idle state). Two considerations hold. Firstly, the applications authenticate the user and check if any content has to be updated. Note how SkyDrive requires about 150 kB in total, 4 times more than others. This happens because the application contacts many Microsoft Live servers during login. Secondly, once login is completed, the applications keep exchanging data with the cloud. Wuala is the most silent, polling servers every 5 minutes on average – *i.e.,* equivalent background traffic of about 60 b/s. Google Drive follows close, with a lightweight 40 s polling interval (42 b/s). Dropbox and SkyDrive use intervals close to 1 minute (82 b/s and 32 b/s, respectively).

Amazon Cloud Drive is completely different: polling is done every 15 s, each time opening a new HTTPS connection. This notification strategy consumes 6 kb/s – *i.e.,* about 65 MB per day! This information is relevant to users with bandwidth constraints and to the system: a user connected to prepaid 3G/4G

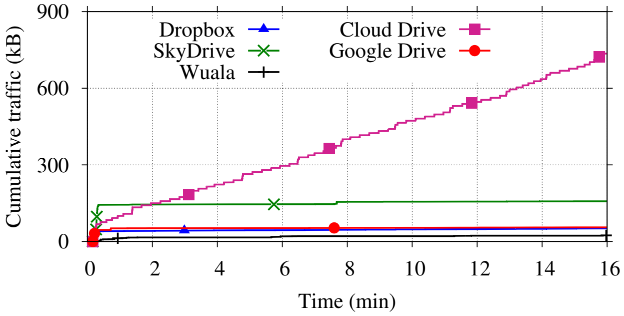


Figure 5.2: Background traffic while idle.

networks in the Netherlands in 2013, for instance, would have to pay around 0,65 euros per day without exchanging any files; similarly, 1 million users would generate approximately 6 Gb/s of signaling traffic alone! As the results for other providers demonstrate, such design is not optimal and seems indeed possible to be improved.

These findings show the relevance of protocol design. In particular, cloud storage services require mechanisms to notify and keep clients updated, periodically producing workload to both servers and the network. Even if each of such notifications may by relatively small when compared to users' files, some design choices can result in surprisingly high costs to clients and to the system.

### 5.2.2 Data Centers

Next, we analyze data center topologies and locations. These aspects influence cloud storage performance primarily because of the network latency, as Section 5.5 will analyze. Moreover, storing data in the cloud has privacy and social implications (see Chapter 1), because cloud providers and, possibly, authorities can access people's data without being noticed, if data are not previously encrypted by users. Knowing data center locations is, therefore, important for helping people to understand possible issues of storing data in the cloud.

Figure 5.3 depicts the identified locations of four services. Dropbox uses own servers (in the San Jose area) for client management, while storage servers are committed to Amazon in Northern Virginia. Cloud Drive uses three AWS data centers: two are used for both storage and control (in Ireland and Northern Virginia); a third one is used for storage only (in Oregon). SkyDrive relies on Microsoft's data centers in the Seattle area (for storage) and Southern Virginia (for storage and control). We also identified a destination in Singapore (for control only). Not surprisingly, most data centers are located in the U.S.

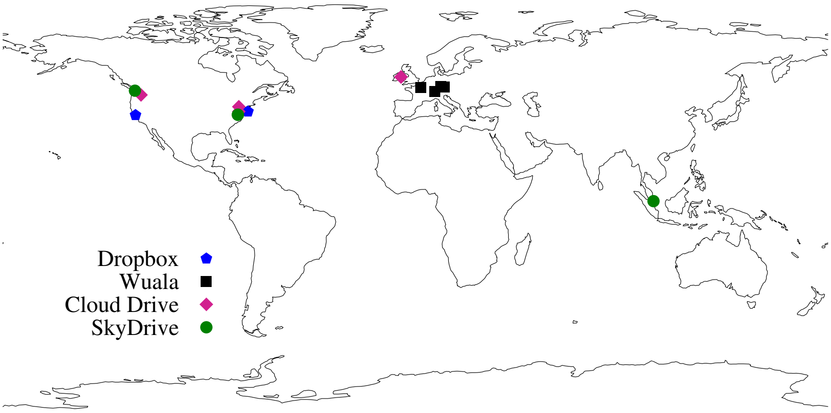


Figure 5.3: Data center locations. Points around Virginia (U.S.) are slightly displaced for improving visualization.

Wuala data centers instead are located in Europe: two in the Nuremberg area (Germany), one in Zurich (Switzerland) and a fourth in Northern France. None is owned by Wuala. All these services follow a centralized design where clients contact the servers directly using the public Internet, as expected.

Google Drive follows a different approach: TCP connections are terminated at the closest Google's *edge node*, from where the traffic is routed to the actual storage/control data center using the private Google network. Figure 5.4 shows the locations identified in our experiments.[4] Overall, more than 100 different entry points have been located. Such architecture allows to reduce client-server RTT and to offload storage traffic from the public Internet. Performance implications are discussed in Section 5.5.

## 5.3 Crowd-Sourced Files

Our results in Chapter 4, obtained using passive flow measurements, provided evidences about the characteristics of files stored in the cloud. For example, our study revealed that:

1. Typical Dropbox flows exchange few bytes of payload. This fact, together with our knowledge of the Dropbox protocol, suggests that files stored in the service are generally small;

[4] Our results match with Google's points of presence [73]. Understanding how Google manages traffic inside its network is outside the scope of this thesis.

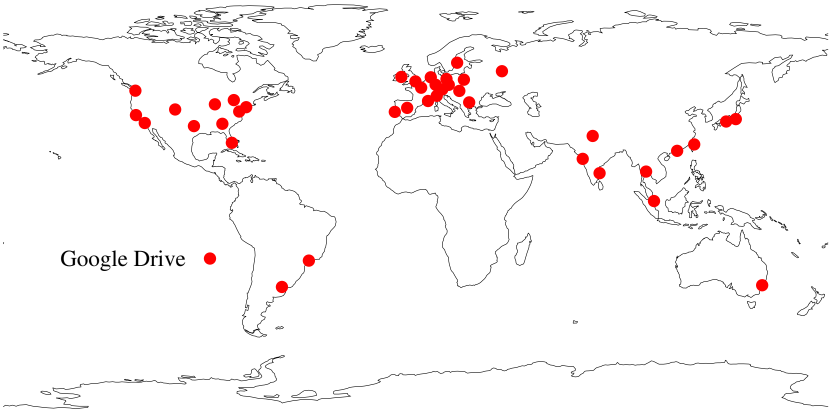


Figure 5.4: Google Drive's edge nodes.

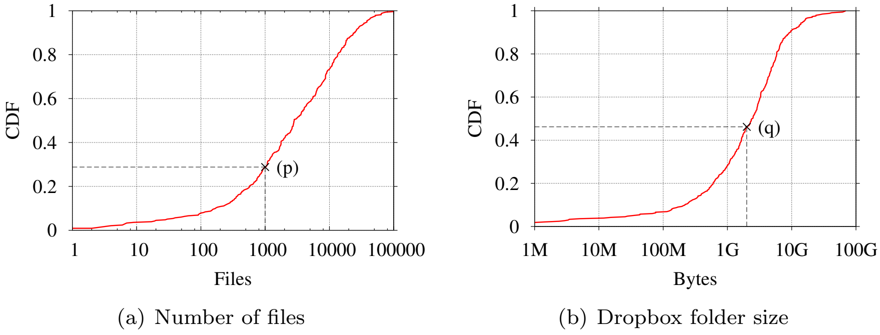
2. A significant portion of Dropbox flows carry several chunks of data, suggesting that files are often added in groups to the service;
3. Replication is common among different users, because of shared folders.

This section summarizes the outcome of our data collection with volunteering Dropbox users, which has been performed with two goals. Firstly, we want to verify and reinforce our conjectures of Chapter 4, by directly reading and analyzing the files of a sample of Dropbox users. Secondly, we want to extend our knowledge about the files stored in the cloud with information that cannot be observed in passive flow measurements, such as file meta-data.

Section 5.3.1 gives an overview of our data collection and the obtained dataset. Sections 5.3.2–5.3.5 present characteristics of the files in volunteers' folders, including file size distributions, replication, content types and file creation dynamics. Finally, Section 5.3.6 lists client capabilities that providers can implement to explore the identified file characteristics and improve performance.

### 5.3.1 Dataset Summary

Our data collection has been performed by means of a standalone application, publicized via mailing lists and social networks. The call for participation has been distributed for 2 months (in 2013). Our program has been executed by 333 unique Dropbox users, who have around 3 million files in Dropbox, totalizing 1.38 TB of data. Most participants are from Brazil (45 %), Europe (40 %) and the U.S. (7 %). The collected meta-data include content type, file size, last modification time, encrypted name and MD5 hashes of initial and final 8 kB of



(a) Number of files

(b) Dropbox folder size

Figure 5.5: Usage per volunteer (note the logarithmic $x$-axis).

each file. It should be noted that, while our sample is certainly biased toward few geographic regions and people working in the academia, it provides strong indications to support our conjectures of Chapter 4 and, thus, define realistic benchmarks to study cloud storage services. Moreover, this is the first work to characterize files stored in Dropbox. Our results are an initial reference to other measurement studies, which might extend the analysis in the future.[5]

Figure 5.5 summarizes the dataset by presenting the status of volunteers' Dropbox folders. Figure 5.5(a) shows the CDF of the total number of files of each volunteer, whereas Figure 5.5(b) shows the CDF of volunteers' used space. We can conclude from these figures that most volunteers synchronize a high number of files, with 70 % having more than 1,000 files – see point (p) in Figure 5.5(a). Usage is generally close to the initial limit given for free by Dropbox – around 2 GB, marked by point (q) in Figure 5.5(b), although a significant portion of volunteers is already above such limit.

### 5.3.2 File Sizes

Figure 5.6 depicts the CDF of file sizes considering all volunteers. This figure leads to the first important conclusion for defining our benchmarks, reinforcing our conjecture of the previous chapter: the vast majority of files in volunteers' folders are very small – *e.g.,* around 60 % of the files have 10 kB or less – see point (p) in Figure 5.6. Moreover, this finding raises questions about network overheads, since most storage communication is encrypted and TLS/SSL handshakes involve certificate exchanges that are of a few kilobytes too.

[5] All data discussed in this section is available to the public at `http://www.simpleweb.org/wiki/Dropbox_Crawler`.

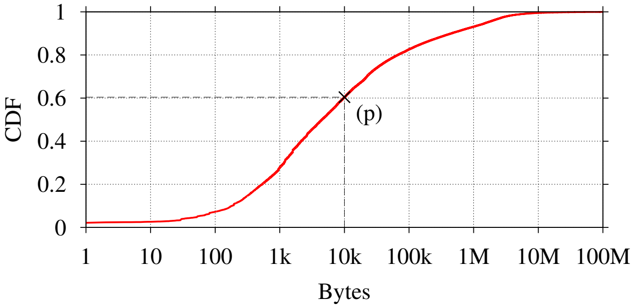


Figure 5.6: Overall file size (note the logarithmic $x$-axis).

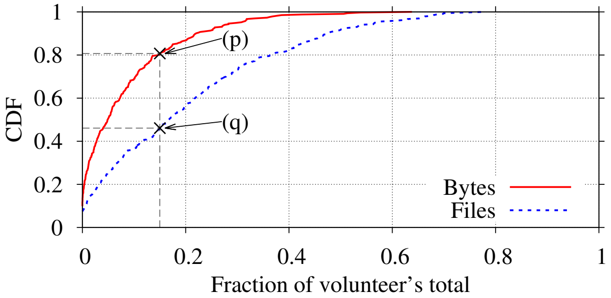


Figure 5.7: CDF of the number of volunteers per fraction of replicas.

Figure 5.6 also shows that a small but non-negligible percentage of files are over 1 MB in size. The distribution is long-tailed, with the biggest file in our sample having more than 5.5 GB, for instance. Therefore, our second conclusion is that cloud storage services need to be able to efficiently handle huge files, although less frequently.

### 5.3.3 Replicas

Figure 5.7 depicts the CDF of the number of volunteers according to the fraction of replicas in their folders. The figure reports only repeated content – *i.e.*, whenever two or more files with the same content are identified, all bytes/files are accounted, except for the first occurrence. For speeding up the data collection, files are considered replicas if they have the same size and share the first and last 8 kB. Manual analyses reveal that most replicas also share names, but are in distinct folders, suggesting that our approach provides a good approximation.

Table 5.2: Percentage of bytes and files on top-10 content-types. The line in bold marks the only content type that is surely compressible.

| Content-type | % of bytes | % of files |
|---|---|---|
| image/jpeg | 19.6 | 8.0 |
| application/pdf | 14.5 | 6.2 |
| application/octet-stream | 12.4 | 12.7 |
| application/zip | 8.3 | 2.2 |
| **text/plain** | **7.1** | **30.5** |
| video/mp4 | 5.5 | < 0.1 |
| video/quicktime | 3.4 | < 0.1 |
| video/3gpp | 3.4 | < 0.1 |
| video/x-msvideo | 2.9 | < 0.1 |
| application/x-iso9660-image | 2.7 | < 0.1 |
| Others | 20.2 | 40.4 |

Figure 5.7 shows that content replication is significant. The point (p) in the figure marks the percentage of volunteers (*i.e.,* 80 %) that have up to 15 % of replicated files. This number implies that the remaining 20 % of volunteers have at least 15 % of bytes in replicas. The percentage is even higher for the number of files: more than 50 % of the volunteers have at least 15 % of replicated files – see mark (q) in Figure 5.7. Overall, 42 % of the files (*i.e.,* 1.3 million files) and 14 % of the bytes (*i.e.,* around 204 GB) in our crowd-sourced sample are replicas. These numbers not only confirm our conjecture in Chapter 4 that content duplication among different users is frequent (because of shared folders), but also show that duplication is common *inside* folders of individual users. Therefore, cloud storage services have significant room for saving bandwidth and storage by skipping transmitting and storing duplicated content. The next section will discuss how some cloud providers avoid submitting replicas by implementing *client-side deduplication.*

### 5.3.4 Content Types

The content type is determined by means of the `libmagic` [125]. Table 5.2 shows the top-10 identified content types. Formats in Table 5.2 account for around 80 % of the bytes in our sample. Around 12 % of the bytes are in files that cannot be precisely identified by the `libmagic` and are reported as *application/octet-stream* in the table. A variety of file extensions are noticeable in this group, pointing mostly to executable and multimedia files.

Some inconsistencies also appear on the content subtype, such as files with `mp4` extension identified as either *video/mp4* or *video/3gpp*. This happens be-

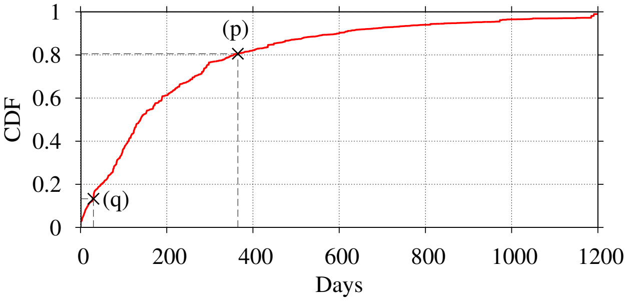


Figure 5.8: File age since last modification time.

cause our crowd-sourcing application first tries to determine the content type using the initial 8 kB of a file, for performance considerations. The complete file is read only when this first attempt fails. The following conclusions, however, are not affected by these inconsistencies.

Our results show that formats that are surely incompressible (*e.g., application/zip*) comprise more than 60 % of the bytes in total. However, *text/plain* and some other compressible formats are non-negligible: around 9 % of the bytes and more than 30 % of the files. These percentages show that incompressible files are smaller than the compressible ones, as it could be expected, since movies and images are usually larger than documents and text files, for example. More importantly, these results suggest that cloud storage services should compress data before transmission, but taking file types into account to avoid wasting of resources.

### 5.3.5 File Creation and Aging

Next, we want to understand how files are created and whether they are changed often in the cloud. Figure 5.8 shows the distribution of file ages, calculated as the difference between file last modification times and the time in the moment of our data capture. Focusing on mark (p) in Figure 5.8, note that almost 80 % of the volunteers' files have been modified less than 1 year before our data collection. The figure is smooth, with only 14 % of the files modified in the last month before our capture (point (q) in the figure). These results suggest that files are relatively young in the service, but not constantly changed. We cannot make any definitive assertion about how files evolve over time, however, since our application collects a single snapshot.

Last modification times also allow us to speculate about how files are created and modified in the service. Section 4.4.3 already showed that a significant

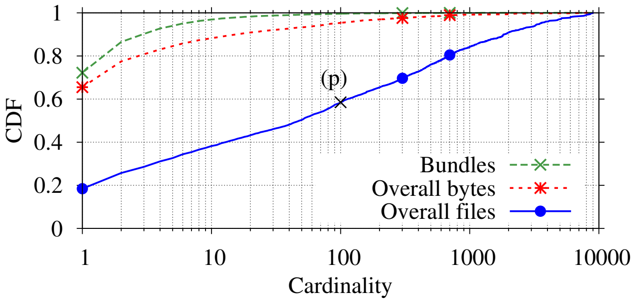


Figure 5.9: Bundles – files of a volunteer are grouped by last modification time.

portion of Dropbox *flows* in the network carries several chunks of data, thus suggesting that files are often added (or modified) in *bundles*. Our flow measurements, however, do not provide insights on the files inside the bundles.

We perform this analysis by grouping the files of each volunteer by last modification time – arbitrary time bins of 10 s are used for illustration, with smaller bins leading to similar conclusions. Three properties of the obtained bundles are analyzed: (i) the size of the bundles (*i.e.*, their cardinality); (ii) the number of bytes that are in bundles of a specific cardinality; and (iii) the number of files that are in bundles of a specific cardinality. Figure 5.9 shows the CDF of these three variables (note the logarithmic $x$-axis). Focusing on the line representing the CDF of the number of bundles per cardinality, note that 70 % of the created bundles have just 1 file. However, the tail of the distribution is long: the remaining bundles have up to 10,000 files. The distribution of bytes shows that single-file bundles include around 70 % of the data.

More interestingly, although 70 % of the bundles have just a single file, most files in our sample are in the remaining 30 % of the bundles. Figure 5.9 depicts that only 20 % of the files are in single-file bundles. Moreover, 40 % of the files are in bundles of at least 100 (mark (p) in the figure). That is, most files in our sample are small (see Figure 5.6) and multiple small files are frequently added or modified at once. These findings imply that cloud storage services should be prepared to handle bundles containing lots of small files.

### 5.3.6 Implications

Our study gives an indication about the type of workload that cloud storage applications need to handle. Similar workloads have already been faced by traditional networked file systems, which often implement special client capabilities to optimize storage usage and to speed up transfers in specific scenarios [114, 140]. These client capabilities include the adoption of:

- *Bundling*, *i.e.,* the transmission of multiple small files as a single object;
- *Chunking*, *i.e.,* splitting content into a maximum size data unit;
- *Compression*;
- *Client-side deduplication*, *i.e.,* avoiding re-transmitting content already available on servers;
- *Delta encoding*, *i.e.,* the transmission of only modified portions of a file.

The next section will investigate whether the selected providers implement these capabilities in their clients, by performing a series of active experiments in our testbed. After that, Section 5.5 discusses performance implications of the different designs.

## 5.4 Cloud Service Capabilities

### 5.4.1 Bundling

Section 5.3 showed that most files in Dropbox are small and added in groups. When a batch of files needs to be transferred, files could be bundled and pipelined, such that both transmission latency and control overhead are reduced. Our benchmark to check how services handle batches of files consists of 4 upload sets, each containing exactly the same number of bytes, which are split into 1, 10, 100 or 1000 files, respectively.

These experiments reveal a variety of synchronization strategies. Google Drive and Cloud Drive open one separate TCP (and TLS/SSL) connection for each file. Considering management, Cloud Drive opens 3 TCP and TLS/SSL control connections per file operation. Figure 5.10 shows the number of TCP `SYN` packets observed when Google Drive and Cloud Drive have to store 100 files of 10 kB each: 100 and 400 connections are opened respectively, requiring 30 s and 55 s to complete the upload. Section 5.5 will confirm that such design strongly limits the client performance when several files have to be exchanged, owing to TCP and TLS/SSL negotiations.

Other services reuse TCP connections. However, SkyDrive and Wuala submit files sequentially, waiting for application layer acknowledgments between each file upload. This can be determined by counting packet bursts, which are proportional to the number of files in our experiments. The same strategy is seen in older versions of Dropbox (see Chapter 4). Dropbox, however, implements a file-bundling strategy since its version 1.4.0.

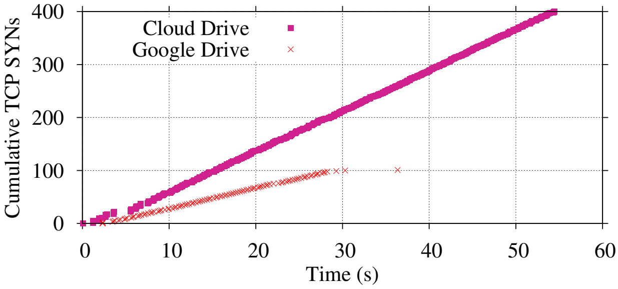


Figure 5.10: Uploading 100 files of 10 kB. Other services are not shown since a single connection is normally used to store batches of files.

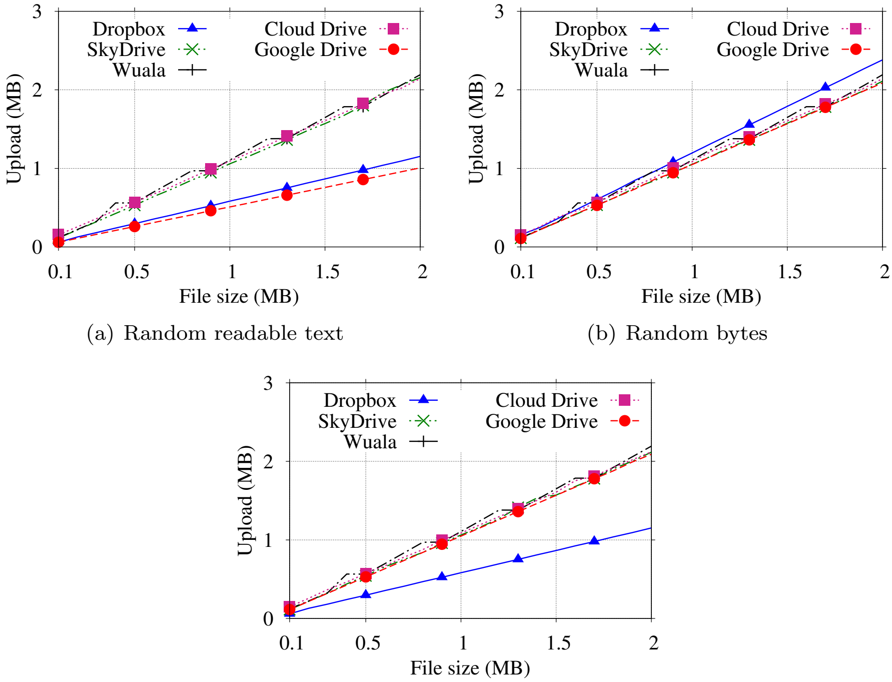
### 5.4.2 Chunking

Our next test aims at understanding how the services process large files. By monitoring throughput during the upload of files differing in size, we determine whether files are exchanged as single objects (no pause during the upload), or split into *chunks*, each delimited by a pause. Our experiments show that only Cloud Drive surely does not perform chunking. In fact, Google Drive uses 8 MB chunks while Dropbox uses 4 MB chunks. SkyDrive and Wuala apparently vary chunk sizes, since different throughput patterns can be identified over different experiment rounds.

Chunking seems advantageous because it simplifies upload recovery in case of failures: partial submission becomes easier to be implemented, benefiting users connected to slow networks, for example.

### 5.4.3 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 benchmark the compression capability with two distinct file sets. The first set (Figure 5.11(a)) is made of highly compressible text files (sizes from 100 kB to 2 MB). Files in the second set (Figure 5.11(b)) contain pure random bytes so that compression is ineffective. Figure 5.11(a) 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 benchmark size. Figure 5.11(b) confirms that Dropbox has the highest overhead in this scenario. This overhead, which is mostly caused by the Dropbox control protocol, will be further discussed in Section 5.5.



(a) Random readable text

(b) Random bytes

(c) Fake JPEGs

Figure 5.11: Bytes uploaded during the compression test.

Naturally, compression is advantageous only for some file types. Compression has a negligible or negative impact when already compressed files are going to be transmitted. Results in Section 5.3 show that incompressible formats are indeed the majority, although a significant portion of compressible files are stored in the cloud. 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, as it is implemented in the `libmagic` [125]. We check whether Google Drive and Dropbox implement smart policies by creating fake JPEGs – *i.e.,* files with JPEG extension and JPEG headers, but actually filled with text that can be compressed. Figure 5.11(c) reveals that Google Drive identifies JPEG content and avoids compression. Dropbox instead compresses all files independently of contents and extensions. Hence, in case of true JPEG files, resources are wasted.

### 5.4.4 Client-Side Deduplication

Server data deduplication eliminates replicas on the storage server. In case a file is already present, replicas in the client folder can be identified to save upload capacity too. This can be accomplished by calculating a hash value using the file content (*e.g.,* SHA256 is used by Dropbox [113]). The hash value is sent to servers prior to submitting the complete file. Servers can check whether the hash is already stored in the system and skip the upload of repeated files. Our crowd-sourced sample shows that file replicas are common inside users' folders and, therefore, implementing client-side deduplication can save a significant amount of bandwidth.

To check whether client-side deduplication is implemented, we design the following experiment:

1. A file with random content is created in an arbitrary folder – since this is the first copy of the file, the full content is transferred in the network;

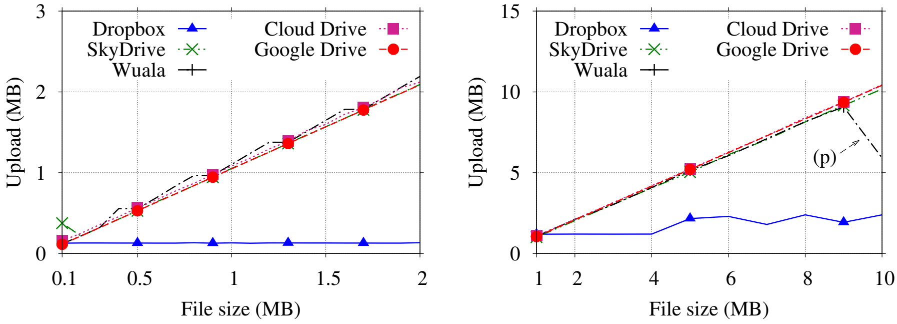
2. The same random content is used to create a replica of the file with a different name in a second folder – assuming hashes are used to identify replicas, only meta-data should be transferred, and not the complete file again;

3. 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 the copying of files to different folders;

4. After all copies are deleted, the original file is placed back – this last step determines whether deduplication still works after all copies of a file are deleted from the local folder.

Results allow to conclude that only Dropbox and Wuala implement deduplication. All other services have to upload the same data even if it is readily available at the storage server. Interestingly, Dropbox and Wuala can identify copies of users' files even after they are deleted and later restored. In the case of Wuala, deduplication is compatible with local encryption – *i.e.,* two identical files generate two identical encrypted versions.

Finally, Dropbox used to implement inter-user deduplication [113]. Inter-user deduplication allows clients 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, if no proof-of-ownership method is employed [81]. By manually performing extra experiments with different users, we conclude that none of the services implement inter-user deduplication anymore.



(a) Append (b) Random

Figure 5.12: Delta encoding tests. Note the $x$-axes.

### 5.4.5 Delta Encoding

Delta encoding is a specialized compression technique that calculates file differences among two copies, allowing the transmission of only the modifications between revisions. To verify which services implement delta encoding, a sequence of changes is generated on a file such that a portion of content is added/changed at each iteration. Three cases are considered: new data added/changed (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 [140] are implemented. In all cases, the modified file replaces its old copy.

Figure 5.12(a) shows the number of bytes uploaded at each step of the experiment in which data are added at the end of the file (*i.e.,* appended). File sizes have been chosen up to 2 MB in this case, and 100 kB are appended at each iteration – *e.g.,* all content in the 1 MB file plus 100 kB of random bytes are used to create the 1.1 MB file at the next iteration of the experiment. Larger files are instead considered in Figure 5.12(b) to highlight the combined effects with chunking and deduplication. In this second case, files of up to 10 MB are generated, and 1 MB of content is added at a random position within the file at each iteration. Other experiments (*e.g.,* adding content at the beginning of files) are not shown, since similar conclusions are obtained.

We see that only Dropbox implements delta encoding – *e.g.,* the volume of uploaded data in Figure 5.12(a) corresponds to the actual part that has been modified. The use of chunking, however, may increase the sent traffic in certain circumstances. For example, focusing on Dropbox in Figure 5.12(b), observe that the amount of traffic increases when files are bigger than the Dropbox 4 MB-long chunk. This happens because the original content may be shifted by

Table 5.3: Summary of the capabilities implemented in each service.

| | Dropbox | SkyDrive | Wuala | Google Drive | Cloud Drive |
|---|---|---|---|---|---|
| Bundling | **yes** | no | no | no | no |
| Chunking | 4 MB | variable | variable | 8 MB | no |
| Compression | always | never | never | **smart** | never |
| Deduplication | **yes** | no | **yes** | no | no |
| Delta encoding | **yes** | no | no | no | no |

the new bytes added at a random position, changing several chunks at once. As such, the volume of data to be transmitted is larger than the added data.

Wuala does not implement delta encoding. However, deduplication prevents the client from uploading chunks not affected by the change. This can be seen in Figure 5.12(b) – see mark (p): when 1 MB is added at a random offset forming a 10 MB file, only some chunks are modified, and thus uploaded.

Delta encoding may have a positive impact on storage performance if files are constantly changed as, for instance, when people perform iterative work on synchronized folders. On the other hand, the storage of static content is not affected by this feature. Our crowd-sourced sample suggests that most files are in formats that, normally, are not incrementally edited, such as JPEG images. However, our sample is neither informative nor representative enough to estimate the overall impact of delta encoding, since only a single snapshot has been taken from a limited number of users. This analysis requires longitudinal data about the evolution of file systems, and might be performed in future work.

### 5.4.6 Summary

Table 5.3 summarizes the capabilities of each service, by showing: (i) whether the service implements bundling, deduplication and delta encoding; (ii) the used threshold to split files in chunks (or whether the threshold is *variable*); and (iii) whether content is compressed *always*, *never* or based on file formats (*i.e., smart*). We can see that Dropbox has the most sophisticated client from the point of view of features to enhance synchronization speed. Wuala, Google Drive and SkyDrive come next, implementing some capabilities. Finally, Cloud Drive has the most simplistic client, as none of the checked capabilities have been implemented.

Table 5.4: Benchmarks to assess client performance.

| Binary files (random bytes) | | | Plain text | | |
|---|---|---|---|---|---|
| Set | Files | Size | Set | Files | Size |
| 1 | 1 | 100 kB | 5 | 1 | 100 kB |
| 2 | 1 | 1 MB | 6 | 1 | 1 MB |
| 3 | 10 | 100 kB | 7 | 10 | 100 kB |
| 4 | 100 | 10 kB | 8 | 100 | 10 kB |

## 5.5 Client Performance

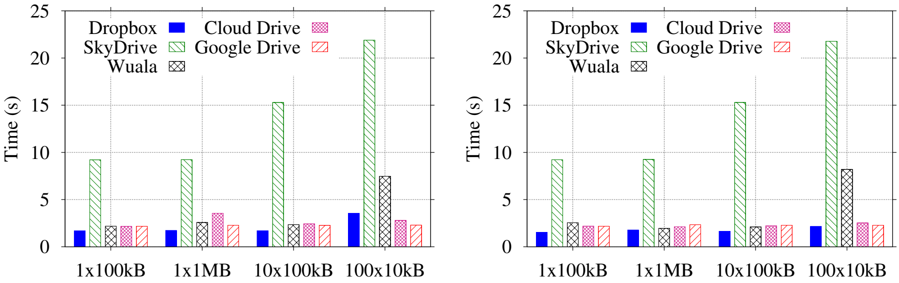
After documenting the architecture and capabilities of each service, we quantify the impact of design choices on performance. Based on the information described in Section 5.3, we develop 8 benchmarks varying (i) the number of files; (ii) the file sizes; and (iii) the file formats. Table 5.4 lists our benchmark sets. All files in the sets are created at run-time by our testing application. We do not include benchmarks with files that are constantly changed or file replicas because most services do not implement any functionality targeting these cases. Naturally, results in these scenarios would be identical for the services that do not implement specific capabilities.

All experiments are executed precisely in the same controlled 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 24 times per service, allowing at least 5 min between experiment rounds to avoid creating abnormal workload to servers and the network. Furthermore, to prevent our results from being affected by *systematic sampling* bias [127], we test a different benchmark set (see Table 5.4) at each experiment round and execute other tasks that last for a variable length of time between the rounds, such as the cleaning of the client folders. Therefore, the time between two rounds of a single benchmark is not deterministic. Synchronization startup, upload duration and protocol overhead are discussed in the following.

### 5.5.1 Synchronization Startup

We evaluate how much time each service needs before synchronization starts. This metric could reveal whether implementing advanced capabilities increases the initial synchronization delay. The metric is computed from the moment files start being modified in a test computer (see step 1 in Figure 5.1) until the first storage flow is observed in the network (see step 2 in Figure 5.1).[6]

[6] The metric also includes a small delay of our application to start sending files to the test computer. This artifact can be ignored, however, since all experiments are equally affected.



(a) Random binary files (b) Plain text files

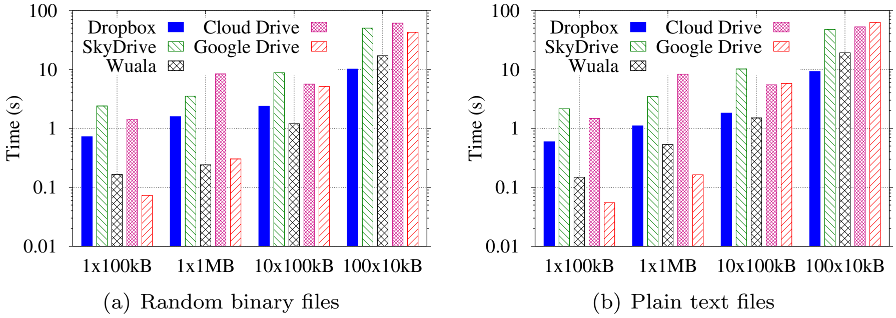
Figure 5.13: Average synchronization start up time.

Table 5.5: Averages and standard deviations of synchronization start up times (in seconds). Numbers are truncated to 2 decimal digits.

| | Test | Dropbox | SkyDrive | Wuala | Cloud D. | Google D. |
|---|---|---|---|---|---|---|
| Binary | 1 x 100 kB | 1.69 (0.28) | 9.21 (0.02) | 2.19 (1.06) | 2.17 (0.11) | 2.17 (0.02) |
| | 1 x 1 MB | 1.74 (0.14) | 9.24 (0.03) | 2.57 (0.97) | 3.55 (6.56) | 2.28 (0.05) |
| | 10 x 100 kB | 1.71 (0.13) | 15.29 (0.04) | 2.35 (0.85) | 2.43 (1.02) | 2.27 (0.03) |
| | 100 x 10 kB | 3.56 (7.91) | 21.89 (0.56) | 7.47 (4.50) | 2.79 (0.47) | 2.30 (0.06) |
| Text | 1 x 100 kB | 1.54 (0.13) | 9.22 (0.03) | 2.55 (1.41) | 2.19 (0.10) | 2.18 (0.02) |
| | 1 x 1 MB | 1.78 (0.18) | 9.25 (0.03) | 1.95 (1.11) | 2.12 (0.14) | 2.34 (0.05) |
| | 10 x 100 kB | 1.64 (0.21) | 15.29 (0.03) | 2.10 (1.05) | 2.22 (0.25) | 2.28 (0.06) |
| | 100 x 10 kB | 2.17 (0.79) | 21.77 (0.06) | 8.20 (3.41) | 2.53 (0.19) | 2.28 (0.03) |

Figure 5.13(a) depicts results in 4 scenarios using binary files. Dropbox is the fastest service to start synchronizing single files. Its bundling strategy, however, slightly delays the start up with multiple files. As we will show next, such strategy pays back in total upload time. SkyDrive is by far the slowest, waiting at least 9 s before starting submitting files. The root cause of this delay is unclear, since the SkyDrive client does not report any activity during this period. Moreover, SkyDrive gets slower as batches increase in size, taking more than 20 s to start sending 100 files of 10 kB. Wuala also increases its startup time when multiple files are submitted.

Similar conclusions are obtained with plain text files (see Figure 5.13(b)). Table 5.5 complements the results, by showing averages along with standard deviations. All services generally produce consistent averages for both file formats, with relatively small standard deviations after 24 runs. Some exceptions are seen in the results for Dropbox (*e.g.,* 100 binary files of 10 kB) and Cloud Drive (*e.g.,* 1 binary file of 1 MB). These exceptions are caused by a single execution round



(a) Random binary files

(b) Plain text files

Figure 5.14: Average upload duration (note the logarithmic $y$-axis).

in each case (*i.e.,* an outlier).[7] It is not clear in our results whether the outliers are measurement errors, or whether the distributions are heavy-tailed. Nevertheless, results in Table 5.5 reinforce our conjecture that clients can implement advanced capabilities, without increasing synchronization delays.

### 5.5.2 Upload Duration

We test how long each service takes to complete upload tasks. This is measured as the difference between the first and the last packet with payload seen in any storage flows. Similarly to our measurements in the previous chapter (see Appendix B), we ignore TCP tear-down delays, and control messages sent after the upload is complete.[8]

Figure 5.14 summarizes our results (note the logarithmic scale on the $y$-axis). A mixed figure emerges. When synchronizing single binary files of 100 kB or 1 MB, the distance between our testbed and the data centers dominates the metric. Google Drive (26.49 Mb/s) and Wuala (33.34 Mb/s) are the fastest, since each TCP connection is terminated at data centers nearby our testbed. Dropbox and SkyDrive, on the other hand, are the most impacted services. SkyDrive (160 ms of RTT), for instance, needs almost 4 s to upload a 1 MB file, whereas Google Drive requires only 300 ms (15 ms of RTT).

When multiple files are stored, the client capabilities play a central role. The rightmost bars in Figure 5.14(a) show a striking difference on transfer duration when 100 files of 10 kB are used. Dropbox wins by a factor of 2 because of bundling, topping to 0,8 Mb/s of upload rate. Interestingly, Google Drive's

[7] Interested readers can download all datasets used in this chapter from `http://www.simpleweb.org/wiki/cloud_benchmarks`.

[8] Note that some clients allow users to limit the upload rate manually. This functionality has been disabled if available.

Table 5.6: Averages and standard deviations of upload durations (in seconds), truncated to 2 decimal digits.

| | Test | Dropbox | SkyDrive | Wuala | Cloud D. | Google D. |
|---|---|---|---|---|---|---|
| Binary | 1 x 100 kB | 0.72 (0.08) | 2.38 (0.79) | 0.16 (0.09) | 1.43 (0.09) | 0.07 (0.00) |
| | 1 x 1 MB | 1.59 (0.16) | 3.51 (1.00) | 0.24 (0.05) | 8.36 (0.39) | 0.30 (0.01) |
| | 10 x 100 kB | 2.38 (0.30) | 8.79 (2.48) | 1.20 (0.39) | 5.62 (4.14) | 5.11 (1.22) |
| | 100 x 10 kB | 10.12 (3.63) | 49.98 (16.92) | 17.04 (4.08) | 60.74 (34.78) | 42.32 (10.50) |
| Text | 1 x 100 kB | 0.59 (0.05) | 2.15 (0.41) | 0.15 (0.00) | 1.47 (0.09) | 0.05 (0.00) |
| | 1 x 1 MB | 1.10 (0.11) | 3.47 (0.81) | 0.53 (0.81) | 8.27 (0.43) | 0.16 (0.02) |
| | 10 x 100 kB | 1.82 (0.24) | 10.11 (9.05) | 1.50 (0.60) | 5.47 (1.23) | 5.74 (2.22) |
| | 100 x 10 kB | 9.21 (3.56) | 47.28 (6.41) | 19.10 (10.09) | 52.36 (1.33) | 62.31 (8.92) |

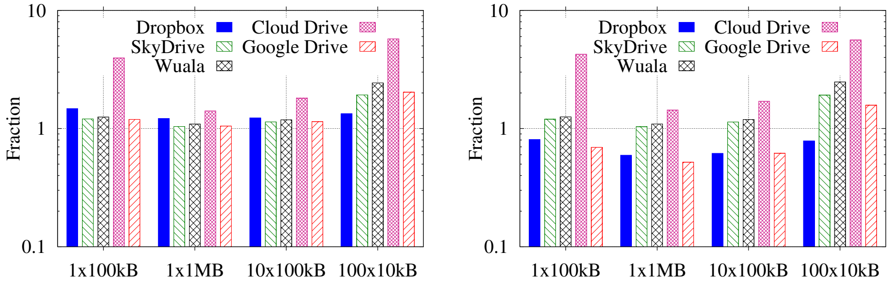
advantage due to its distributed topology is completely canceled by the usage of separate TCP and TLS/SSL connections per file. It takes 42 s on average – *i.e.,* 189 kb/s. Other services are also penalized by their lack of bundling, with Cloud Drive taking about 60 s (132 kb/s) to complete some tests.

Figure 5.14(b) and Table 5.6 complement the analysis. As for the synchronization startup, outliers increase the standard deviation in some scenarios, *e.g.,* when Cloud Drive uploads 10 text files of 100 kB and when SkyDrive uploads 10 binary files of 100 kB. All results with 100 files of 100 kB have high standard deviations, which allow us to conclude that the transfer times of all five services are very unstable when synchronizing lots of small files.

Dropbox is slightly faster with plain text files because of compression, although network latency still dominates the metric. Google Drive profits from compression as well, when sending single plain text files. However, the service is unexpectedly the slowest when multiple small text files are added, showing that its smart compression advantage is also canceled by the client design.

### 5.5.3 Protocol Overhead

Finally, we evaluate protocol overhead. Figure 5.15(a) shows the overhead of each service, calculated as the ratio between *the total storage and control traffic* over *the benchmark size.* Note that the use of compression may lead to ratios smaller than 1. We see in the figure that all services have a significant overhead when small binary files are synchronized. Cloud Drive presents the highest overhead because of the several control flows opened for every file transfer (see Figure 5.2). Dropbox exhibits the highest overhead among the remaining services, possibly owing to the signaling cost of implementing its advanced capabilities: on average, it sends 1.47 times more bytes than the benchmark size when uploading a single 100 kB binary file and 1.22 times more bytes when uploading a single 1 MB binary file – see the exact numbers on Table 5.7.



(a) Random binary files (b) Plain text files

Figure 5.15: Average overhead. The logarithmic $y$-axis depicts the ratio of *control and storage traffic* in the network over *the benchmark size.*

Table 5.7: Averages and standard deviations of the overhead ratio, truncated to 2 decimal digits. The table lists the ratio of *control and storage traffic* in the network over *the benchmark size.*

| | Test | Dropbox | SkyDrive | Wuala | Cloud D. | Google D. |
|---|---|---|---|---|---|---|
| Binary | 1 x 100 kB | 1.47 (0.03) | 1.21 (0.05) | 1.25 (0.03) | 3.95 (2.29) | 1.19 (0.01) |
| | 1 x 1 MB | 1.22 (0.01) | 1.04 (0.02) | 1.09 (0.00) | 1.41 (0.30) | 1.05 (0.00) |
| | 10 x 100 kB | 1.23 (0.00) | 1.14 (0.03) | 1.19 (0.05) | 1.80 (0.40) | 1.14 (0.00) |
| | 100 x 10 kB | 1.34 (0.01) | 1.92 (0.37) | 2.43 (0.26) | 5.74 (0.26) | 2.03 (0.08) |
| Text | 1 x 100 kB | 0.81 (0.02) | 1.20 (0.02) | 1.25 (0.03) | 4.26 (2.51) | 0.69 (0.04) |
| | 1 x 1 MB | 0.60 (0.00) | 1.03 (0.01) | 1.09 (0.00) | 1.43 (0.30) | 0.52 (0.00) |
| | 10 x 100 kB | 0.62 (0.01) | 1.14 (0.02) | 1.19 (0.05) | 1.70 (0.32) | 0.62 (0.01) |
| | 100 x 10 kB | 0.79 (0.01) | 1.92 (0.33) | 2.48 (0.31) | 5.61 (0.07) | 1.57 (0.04) |

The lack of bundling dramatically increases overhead when multiple small files are uploaded, because the upload of every file requires application layer control traffic to be sent. Google Drive, for instance, exchanges twice as much traffic as the actual data size when sending 100 binary files of 10 kB. Cloud Drive shows even more overhead – *i.e.,* more than 5 MB of data are exchanged to commit 1 MB of binary content. Figure 5.15(b) and Table 5.7 show the results in other scenarios, in which a similar pattern is generally seen. Dropbox and Google Drive are the exceptions when sending plain text files, owing to their compression strategies. Standard deviations are very low, confirming that the amount of overhead is highly constant in all tested scenarios.

## 5.6 Conclusions

The contributions in this chapter are threefold. Firstly, we introduced a methodology to benchmark performance and to check capabilities and system design of cloud storage services. Secondly, we presented a characterization of files stored in such services. Thirdly, we evaluated the implications of design choices on performance by benchmarking five cloud storage providers.

Our analysis shows the relevance of *client capabilities* and *protocol design* to cloud storage services. Dropbox implements most of the checked capabilities, and its sophisticated client clearly boosts performance, although some protocol tweaks seem possible to reduce network overhead. On the other extreme, Cloud Drive bandwidth wastage is an order of magnitude higher than other offerings, and its lack of client capabilities results in performance bottlenecks. SkyDrive shows some performance limitations, while Wuala generally performs well. More importantly, Wuala deploys client-side encryption, and this feature does not seem to affect Wuala synchronization performance.

The analysis of Dropbox, SkyDrive, Cloud Drive and Wuala confirms the role played by *data center placement* in a centralized approach: taking the perspective of European users only, network latency is an important limitation for U.S. centric services, such as Dropbox and SkyDrive. Services deploying data centers nearby our test location, such as Wuala, therefore, have an advantage. Google Drive follows a different approach resulting in a mixed picture: it enjoys the benefits of using Google's capillary infrastructure and private backbone, which reduces network latency and speeds up the system. However, protocols and client features limit performance, especially when multiple files are considered.

Overall, our results are useful to improve the performance of cloud storage applications. Finally, our methodology has been implemented as a set of open source tools and is available to the public at `http://www.simpleweb.org/wiki/cloud_benchmarks`.

# Part III

# Conclusions

CHAPTER 6

# Conclusions

## 6.1 Summary and Findings

Cloud-based services have quickly made the idea of utility computing a reality. For example, when our work started in 2009, cloud storage – used as a case study in this thesis – was largely unknown for Internet users. Dropbox had been officially released less than one year before, and only a few other providers, such as Microsoft, had started experimenting with similar products for this niche market at that time. Four years later, cloud storage has become a pervasive service, already among the most popular Internet applications.

The advantages of the cloud model have attracted more and more customers to the cloud market. However, such migration exposes customers to both dependability problems and privacy threats. This thesis started by arguing that (i) dependability concerns demand new methods to monitor cloud services; (ii) privacy concerns demonstrate the need for new providers to offer stronger privacy and protection against foreign governments. Such *private* and *national* cloud providers need to understand existing cloud services, to compete with international providers in a timely manner. We developed our research around two objectives, associated with four research questions, as follows.

### 6.1.1 Objective 1: Monitoring Cloud Services

Our first objective was to investigate simple and scalable methods for monitoring the performance of cloud services, such that customers could monitor their services easily and independently. We studied whether *flow measurements* provide such monitoring by answering two research questions.

**Research Question 1: Are popular flow-based measurement devices suitable for serving as data source for monitoring cloud services?**

Chapter 2 answered this research question by revisiting the background on flow monitoring and by evaluating the quality of flow data exported by two devices in our production network. The main finding of this analysis is:

- *Measurement errors might be found in widely deployed devices and can turn flow data unusable for advanced applications.*

Two kinds of devices were evaluated in Chapter 2. First, we evaluated a dedicated flow export device, developed by a specialized company (*i.e.,* INVEA-TECH). Our experiments showed that this device works as specified and, thus, it is suitable to our analysis. Second, we evaluated a device from a widely deployed series of switches (Cisco Catalyst 6500), which is equipped with an embedded flow exporter. We showed (Section 2.5) that this device exports flow data with a multitude of errors, including inaccurate time fields and wrongly measured flow attributes. Because of such errors, this device had proven inappropriate for the analysis performed in subsequent chapters.

The lessons learned is that not all flow data sources are appropriate for advanced applications. Therefore, *all* flow export devices must be carefully checked before the deployment of any flow-based application.

**Research Question 2: Are flow measurements suitable to monitor cloud services? What are the limiting factors for such an approach?**

We answered these questions first by proposing a method to monitor the availability of cloud services (see Figure 3.2), which was later extended to monitor the performance of Dropbox (see Section 4.2 and Appendix B). The use of flow measurements to monitor cloud services was then evaluated by means of case studies. We have two main findings from this analysis:

- *Flow measurements are a viable alternative for customers to monitor the performance of cloud services.*

The case studies in Chapter 3 showed that simple application-independent metrics can be easily monitored using flow measurements. We also confirmed that flow-based methods are scalable and non-intrusive. Indeed, without any customizations in our measurement environment, our first case study (Section 3.2) revealed problems in popular cloud services, thus providing evidence of the dependability challenges identified in Chapter 1. Similarly, our second case study (Section 3.3) evaluated the consequences of a case of cyber-demonstration promoted by the *hacktivist* group "*Anonymous*", relying solely on NetFlow data collected in an international backbone. Finally, Chapter 4 showed that complex metrics and systems (*e.g.,* Dropbox) can be monitored as well, provided that flow measurements are carefully tailored and the network is properly instrumented.

- *Proprietary cloud protocols, difficulties for isolating traffic and the need for application-specific metrics might limit the applicability of flow-based methods in certain scenarios.*

Chapter 3 and Chapter 4 revealed limiting factors for the flow-based approach. As an example, the application of our method in Chapter 4 was only possible because a deep understanding on Dropbox internals was gained by means of *reverse-engineering* (Section 4.1). Reverse-engineering of protocols is time-consuming and proprietary protocols are likely to change over time. As another example, the projection of flow measurements into high-level performance metrics (*e.g.,* response times) is clearly an approximation. While we succeeded in finding elements in flow measurements that could be mapped to operations of the encrypted Dropbox protocol (*e.g.,* see Appendix B), there is no guarantee that such *"tricks"* will work for other services.

Therefore, the lessons learned from our second research question indicate that flow measurements are generally suitable to monitor the performance of cloud services. However, the monitoring of each particular service requires specific methods to post-process the flows, which seems feasible only for standardized and well-established protocols.

### 6.1.2 Objective 2: Understanding Cloud Services

Our second objective was to understand how cloud services are implemented, and to understand the implications of their design and usage for the Internet. Two questions were addressed for this objective.

**Research Question 3: What are the typical usage and performance characteristics and bottlenecks of Dropbox?**

We answered this question by presenting the first comprehensive characterization of Dropbox, which is the most popular cloud storage service by the time of writing (see Chapter 4). By means of flow measurements passively collected from operational networks, we showed that Dropbox is a data-hungry application. Moreover, we quantified how Dropbox protocol design and data center locations, together with users' behavior, are reflected on the application network usage. This research question resulted in three key findings:

- *Cloud storage is a new data-intensive application, already among the top applications producing Internet traffic.*

The traffic to Dropbox observed during our measurements is comparable to other widely used Internet applications – *e.g.,* as much as 30 % of YouTube or 4 % of the total traffic in one of the studied networks (Section 4.3). Such numbers were a surprise to us, since the adoption of cloud storage is still in a much earlier stage – *e.g.,* our measurements showed that only 6-12 % of home users rely on cloud storage services.

- *Despite the high popularity of Dropbox, a minority of its users is responsible for most workload.*

Section 4.5 showed that only around 35 % of the users take full advantage of the Dropbox functionality, actively synchronizing devices and sharing several folders. Such users are responsible for more than 80 % of the traffic observed in our measurements. We also noted that around one third of the users completely abandon their clients, rarely exchanging any data during our observations. Other usage patterns relevant for the design of similar services are described in Section 4.5 and include the high popularity of content sharing (*e.g.,* 70 % of users share at least one folder) and the relatively small number of devices per user (*e.g.,* only 30 % of home users have more than one linked device).

- *The sequential submission of files in cloud storage services results in performance bottlenecks. Implementing some form of file bundling or content pipelining is essential.*

We identified that the performance of Dropbox during our measurements was mainly determined by the distance between clients and storage data centers (Section 4.4). The combination of high network latency with the sequential submission of files was shown to penalize Dropbox performance dramatically. This bottleneck was further confirmed by reverse-engineering the Dropbox protocol. We identified as possible solutions the use of file bundling or the introduction of content pipelining. A protocol update performed by Dropbox during our data collection included a file bundling strategy. We showed that such change increased around 65 % the average Dropbox throughput (Section 4.4.5).

These findings are an important asset to new players implementing cloud storage. New providers can profit from our contributions not only to predict the workload their services will have to face, but also to avoid the bottlenecks identified and quantified in our analysis. Moreover, our results are important for the research community, companies and ISPs, to understand and anticipate the impact of a likely massive adoption of cloud storage.

**Research Question 4: How do different providers implement cloud storage services and what are the implications of the design choices for client performance?**

We provided the first thorough analysis of the impact of design choices on cloud storage services. This was achieved by means of a methodology that relies on a testbed, intercept proxies, benchmarking scripts and tools to collect information from volunteers (Section 5.1). Chapter 5 evaluated several aspects of the design and implementation of five popular cloud storage services. We concluded that providers follow different strategies to synchronize files (*e.g.,* see Section 5.2 and Table 5.3). Design differences that seem minor at first can result in performance bottlenecks and surprising costs to both users and the Internet. In particular, Chapter 5 demonstrated the importance of *protocol design*, *client capabilities* and *data center placement* to cloud storage performance:

- *The design of "notification" and "storage" protocols is critical for the performance of cloud storage services.*

Section 5.2 showed that cloud storage services require mechanisms to notify and keep clients updated, periodically producing workload both to servers and to the network. Although it is intuitive that such protocols should produce as little overhead traffic as possible, we showed that some of the most popular providers still deploy inefficient protocols, which significantly impacts client performance and network usage. For example, the bandwidth wastage of Amazon's protocol to notify clients about file changes was found to be one order of magnitude higher than competitors (see Section 5.2.1). Amazon's application can generate up to 65 MB per day even when no files are synchronized. Such wastage is problematic to users with bandwidth constraints, such as those using mobile networks, but also to the system as a whole: 1 million users would generate approximately 6 Gb/s of signaling traffic alone.

- *The typical characteristics of files stored in the cloud allow providers to improve performance by deploying special capabilities at the client-side.*

Results in both Chapter 4 and Section 5.3 showed that files stored in the cloud have particular characteristics that can be exploited to optimize storage usage and speed up transfers. We concluded, for example, that (i) most files are small; (ii) several small files are often added in large groups; and (iii) content duplication is very common. Chapter 5 showed that some providers already take advantage of these particularities. Client capabilities found on Dropbox, such as *client-side deduplication*, *delta encoding* and *bundling* (see Section 5.4), provide not only real performance advantages in terms of latency, but also a significant reduction of Internet traffic. Indeed, Section 5.5 demonstrated that the upload of some file sets takes six times more in some of Dropbox's competitors, wasting three times as much capacity. However, Dropbox also seems to have room for improvements: differences in *control* protocols make Dropbox to transfer 10 % more overhead than its competitors in some common scenarios.

- *Data center placement plays an important role not only in terms of privacy, but also in the overall cloud storage performance.*

Chapter 1 argued that privacy issues will impel the appearance of new national cloud players. Chapter 5 evaluated the location from where popular providers operate and, not surprisingly, most of them are centralized in the U.S. Besides the known privacy implications, Section 5.5 evaluated to what extend data center placement impacts performance. We concluded that the upload time of the same files can be reduced by a factor of 11 when data centers are placed closer to users. Interestingly, we benchmarked a service (*i.e.,* Wuala) that deploys both (i) per-user encryption to increase privacy; and (ii) data centers nearby our test location. Results suggest that the privacy-preserving feature does *not* affect synchronization performance, and the favorable data center location improves the overall system performance. Hence, new players seem to have room for providing stronger privacy to users, while also delivering well-performing services.

Overall, the lessons learned are a permanent reference for engineers designing protocols and provisioning data centers for similar services. Results in Chapter 5 contain valuable insights into the bottlenecks resulting from design choices. They will help *private* and *national* cloud providers to create a next generation of cloud storage services.

## 6.2 Contributions

We now provide a list of contributions of this thesis, organized according to the chapters they are presented.

**Chapter 2 – Understanding Flow Data Sources**

- Reviews the background on flow monitoring, including the basic terminology and the effects of varying common flow export parameters.
- Reveals measurement artifacts in widely deployed flow exporters, by means of active experiments and flow data analysis. The uncovered artifacts are related to missing flows and imprecise fields, and can damage flow data permanently, turning the data unusable to any flow-based application.

**Chapter 3 – Monitoring Cloud Services using NetFlow**

- Introduces a simple method to indicate the availability of cloud services using NetFlow. The method is shown to be robust to handle flows measured under different configurations, including sampled and non-sampled NetFlow.

- Presents an open source plug-in for NfSen [78] implementing the method. The tool can be downloaded from `http://www.simpleweb.org/wiki/Cloud_Monitoring`.
- Applies the method to evaluate the use of flows to monitor cloud services. First, a 10-week measurement study gives evidences of availability problems in cloud services. Then, flows collected from an international backbone show the consequences of a cyber-demonstration.

**Chapter 4 – Dropbox Usage and Performance**

- Presents the first comprehensive characterization of cloud storage services, focusing on Dropbox. The characterization details the (proprietary) Dropbox protocol, general traffic characteristics, typical usage scenarios and the performance experienced by end users.
- Highlights practical implications of protocol designs for both users and the Internet. Bottlenecks resulting from design choices are quantified and possible counter-measures are proposed.
- Provides the first large-scale public dataset reporting the activity of *anonymized* Dropbox users. The data have been capture during 42 days in 4 different locations and can be downloaded from `http://www.simpleweb.org/wiki/Dropbox_Traces`.

**Chapter 5 – Comparing Cloud Storage Services**

- Introduces a benchmarking tool for studying cloud storage services. The tool includes scripts for determining the presence of particular capabilities, data center locations etc. Files of several types can be generated at runtime and performance metrics are calculated automatically. The tool is available at `http://www.simpleweb.org/wiki/cloud_benchmarks`.
- Presents a characterization of files stored in Dropbox. The dataset has been obtained from more than 300 volunteers in a crowd-sourcing experiment. This is the first public dataset describing files stored in cloud storage services. The dataset can be downloaded from `http://www.simpleweb.org/wiki/Dropbox_Crawler`.
- Documents how different providers implement cloud storage services, focusing on the underlying design choices of each service.
- Evaluates the consequences of design choices on performance by means of a series of benchmarks, defined according to our crowd-sourced sample.

## 6.3 Future Work

Regarding the development of cloud storage services, we can list three research directions that emerge from our results:

- Our study was completely based on measurements collected either passively from operational networks or actively using our benchmarking tools. Measurement-based studies suffer from some limitations (*e.g.,* sampling bias). New questions raised by our conclusions seem easier to be answered by taking a *model-based approach.* For example, we concluded that some capabilities, such as file bundling, have a significant impact on performance. However, the difficulties in collecting data from geographically dispersed regions disallow us to speculate about how users worldwide perceive these effects. Extending the results with model-based analysis is, therefore, a natural continuation.

- This thesis provided the first dataset of meta-data from files stored in the cloud. Our data collection, however, reached a somehow limited number of participants, most of them with an academic profile. Furthermore, more collection rounds would be necessary to evaluate the evolution of cloud file systems, the advantages of delta encoding etc. Some works have already characterized long-term aspects of traditional file systems [1]. We will pursue in future work a comparison of cloud storage against traditional file systems. Assuming both systems are similar, the latter could be used to extend our results.

- Migrating parts of enterprise file systems (*e.g.,* backups) to the cloud is a hypothesis that we have both heard from system administrators and read in related work [80]. We plan to study the possible impact of such a migration in future work. In particular, we are interested in understanding how the migration would affect the network, and how users would perceive performance once storage systems are (partially or totally) provided from remote data centers.

Other research directions can be listed in relation to the use of flow measurements in general:

- The previously mentioned limiting factors for flow-based methods are natural directions for future work. For example, the automatic identification of network traffic has already been actively researched [21]. Similarly, several works propose methods to automatically reverse-engineer protocols [141] or find associations among traffic flows [91]. The outcomes of such efforts might provide solutions for the problems identified, but not tackled, in this thesis.

APPENDIX A

# Estimating Connection Status using NetFlow

This appendix validates the intermediate steps of the method presented in Chapter 3 (see Figure 3.2). The goal is to show that the method consistently estimates the number of TCP connections in the network using NetFlow records exported in different scenarios. Therefore, we show that the *health index* calculated solely from NetFlow data matches those that would be obtained by other traffic analyzers that make use of full packet traces.

Section A.1 describes the dataset and the validation methodology. After that, we show results of the validation with non-sampled flow data in Section A.2 and with packet-sampled flow data in Section A.3.

## A.1 Dataset and Methodology

The same packet traces used in Chapter 2, in particular, Section 2.4, to illustrate the consequences of varying parameter settings of flow exporters are used in this validation. As in Chapter 2, we use YAF [89] to generate flow records from the packet traces, because we want to test several flow export scenarios. Moreover, for the analyses that focus on a specific service or organization, we rely on the MaxMind GeoIP Organization database [106] and on IP addresses of cloud providers to filter the traffic.

In both the non-sampled and packet-sampled cases, we follow a validation methodology similar to [103]. The status of all TCP connections in our dataset is evaluated by means of (i) a tool that operates directly on the original packet traces (*i.e.,* the ground truth); and (ii) our method, after converting the traces into flow records using YAF. The results of both the packet-based tool and our method are then compared. We use Bro [116] as the ground truth for our comparisons. Bro is a stateful network monitor that contains a module for analyzing TCP connections. Bro classifies terminated TCP connections using the classes listed in Table A.1.[1]

[1] Bro has other classes that should not be reached when all packets are observed. Less than 0.1 % of the connections in our datasets are terminated in those classes owing to packet loss.

Table A.1: Mapping between Bro classes and our definition of health.

| | Meaning | Bro | Freq. (%) | Description |
|---|---|---|---|---|
| Healthy | Ongoing | S1 | 0.9 | Established, but not terminated |
| | | OTH | 0.5 | Midstream traffic (no flags observed) |
| | Closing | S2 | 0.3 | Established, originator attempt to close |
| | | S3 | 0.4 | Established, responder attempt to close |
| | Aborted | RSTO | 20.9 | Established, originator aborted |
| | | RSTR | 5.7 | Established, responder aborted |
| | Complete | SF | 57.6 | Established and terminated |
| Unhealthy | – | S0 | 7.7 | Attempt without reply |
| | | SH | 0.4 | Attempt, followed by FIN from originator |
| | | REJ | 5.2 | Rejected |
| | | RSTOS0 | 0.4 | Attempt, followed by RST from originator |

We map Bro classes into the two classes of our problem (*i.e., healthy* or *unhealthy*) as follows. *Ongoing*, *closing*, *aborted* and *complete* connections are *healthy.* Most connections marked as *ongoing* or *closing* occur because we have captured data for a limited time interval. As such, several connections have been truncated both at the begin and at the end of our capture. Connections marked as *complete* are those with normal establishment and termination. Connections marked as *aborted* are those that have a normal TCP handshake, but are reset by one of the end-points afterward. This is mainly caused by some client applications that intentionally reset connections after exchanging application layer payload, to avoid the TCP termination delays (see [8]). Those are likely to be successful service requests and, therefore, are *healthy* in our context. Finally, the TCP connections classified as *unhealthy* in Table A.1 are only those that could not carry payload at the transport layer. Table A.1 also shows the frequency of each class in our dataset (see column *Freq. (%)*).

## A.2 Non-Sampled Data

### A.2.1 Comparison Methodology

When dealing with non-sampled data, our method aggregates the flow records generated by each TCP connection. The aggregated records are then classified as either *healthy* or *unhealthy.* Assuming these steps are correct, the *health index* calculated by our method will be equivalent to what would be calculated directly from Bro's output.

Both our method and Bro can list the TCP connections labeled with their final classes. A natural way of validating our method is, therefore, by matching its list of connections with Bro's output. The results of both Bro and our method are matched by looking for connections with common keys and the same start times (in seconds). A confusion matrix is then filled. The confusion matrix is a way of presenting results when evaluating classification models [56]. It presents the original number of instances (*i.e.,* connections, in our case) per class versus the total number of instances that are predicted to belong to each class. The diagonal of the matrix contains correctly classified instances, whereas the remaining cells contain the errors per class. Table A.2 shows an example of such a matrix $M$, in a problem composed of $n$ classes.

Table A.2: Confusion matrix $M$ when classifying instances in $n$ classes.

| | Prediction | | | | |
|---|---|---|---|---|---|
| | | $C_1$ | $C_2$ | ... | $C_n$ |
| True Class | $C_1$ | | | ... | |
| | $C_2$ | | | ... | |
| | ... | ... | ... | ... | ... |
| | $C_n$ | | | ... | |

Since our method may output a different number of connections than the ground truth, besides the *healthy* and *unhealthy* classes, an artificial class is used to count *unmatched* connections in the confusion matrix. Several performance metrics can be derived from the confusion matrix $M$ to compare classification models. Three metrics are used in our experiments: *precision* ($P_i$), *recall* ($R_i$), and the, so-called, $F$-*measure* ($F_i$) [56, 143]. Let $\boldsymbol{\omega} = (\mathit{healthy}, \mathit{unhealthy}, \mathit{unmatched})$, *i.e.,* $\boldsymbol{\omega}$ is a vector with the possible output classes in the classification problem. Then, the three performance metrics are defined as a function of $\omega_i$. The precision $P_i$ represents the fraction of instances correctly classified as being of class $\omega_i$:

$$P_i = \frac{M_{ii}}{\sum\limits_{j=1}^{n} M_{ji}} \tag{A.1}$$

The recall $R_i$ is the fraction of instances of class $\omega_i$ that are correctly classified:

$$R_i = \frac{M_{ii}}{\sum\limits_{j=1}^{n} M_{ij}} \tag{A.2}$$

The $F$-measure is a combination of precision and recall that increases faster when both metrics are simultaneously increased:

$$F_i = \frac{P_i R_i}{(1-\alpha)P_i + \alpha R_i}, \qquad 0 \leq \alpha \leq 1, \tag{A.3}$$

where the parameter $\alpha$ weights the importance of each metric in the results. This can be used, for instance, when failing to identify instances of a class has a different cost than making classification mistakes. We calculate $F_i$ using $\alpha = 0.5$, which means that both precision and recall have equal importance. We refer to [143] for a deeper discussion on methods to compare classification models and to [100] for a practical example in another problem domain.

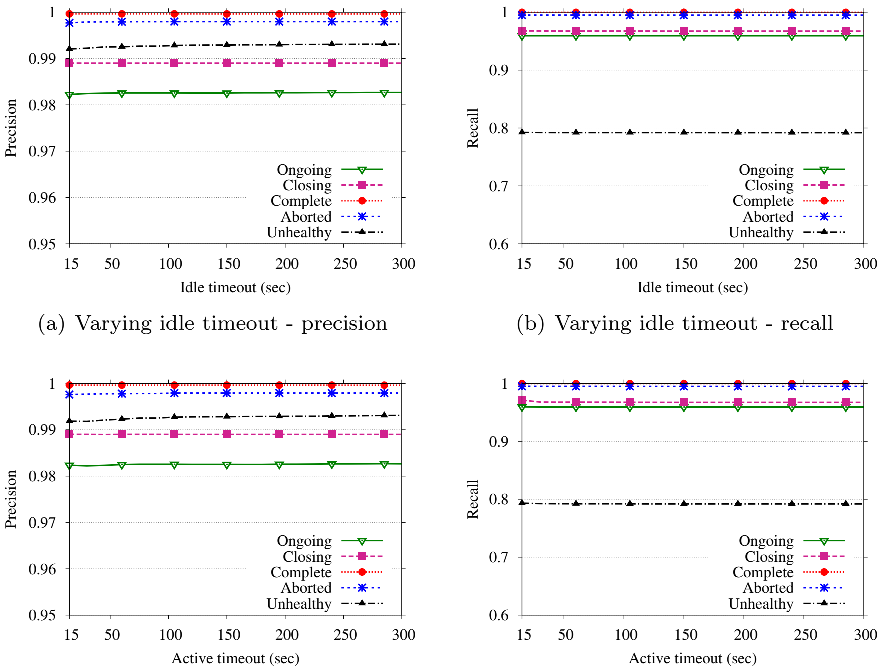
### A.2.2 Tested Scenarios

Two analyses are performed, both using Bro as ground truth. Firstly, because flow exporters may be configured differently and may implement different flow expiration policies (see Chapter 2), the validation is repeated in several export setups. Secondly, since we extend [130], we re-implement the original heuristic and show that our extension indeed improves the results.

Three scenarios are used to check the effects of flow expiration policies:

- **Scenario 1**: Only timeouts expire flow records without any protocol checks. Several flow exporters, such as Vermont [96] and FlowMon [22], implement only this policy because of performance considerations;
- **Scenario 2**: Includes Scenario 1 and flow termination by TCP `RST`/`FIN` packets. Cisco IOS NetFlow and nProbe [36] export flows in this way. However, exporters may implement different heuristics. We use the heuristic of YAF: an `RST` packet in any traffic direction or `FIN` packets followed by acknowledgments in both traffic directions terminate a flow;
- **Scenario 3**: Includes Scenario 2 and the creation of new flows by TCP `SYN` packets. Although not listed in IPFIX standards, this policy would force exporters to separate connections not properly closed into different records. Sequence numbers are evaluated to differentiate `SYN` retransmissions from new connection attempts.

The impact of the idle and the active timeout of flow exporters is also measured for each scenario. This is done by disabling expiration by one of the two timeouts and varying the other one in distinct experiments. Note that both Bro and our method have internal timeout parameters to be set (see Section 3.1.1). The values suggested by Bro for recording connections as accurately as possible (*e.g.,* 2 hours for reporting inactive connections) are used in both cases.



(a) Varying idle timeout - precision

(b) Varying idle timeout - recall

(c) Varying active timeout - precision

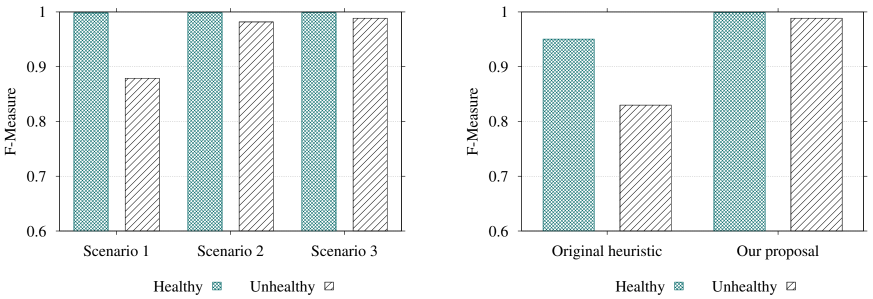
(d) Varying active timeout - recall

Figure A.1: Effects of timeouts on the precision and recall in Scenario 1.

### A.2.3 Results

The impact of timeout parameters in Scenario 1 is depicted in Figure A.1. Figure A.1(a) and A.1(b) show the precision and recall of our method when varying the idle timeout, while Figure A.1(c) and A.1(d) show the results when varying the active timeout. The results for the class *healthy* are shown in a finer granularity (see Table A.1) to illustrate that most errors in this class are from ongoing and closing connections, which should not happen frequently in an on-line deployment.

The figures show that our method is invariant to timeouts. Moreover, the precision is very high for both classes, which means that our method reliably classifies both classes. The recall for the class *unhealthy* is lower, because some connections output by Bro do not have a match in the output of our method. The problem happens because flow exporters may report several connections in a single flow record – *e.g.,* when TCP end-points reuse sockets before a record



(a) Our method in different scenarios

(b) Comparison to [130] in Scenario 3

Figure A.2: Performance of heuristics to reassemble non-sampled flow records.

has been expired by the flow exporter. As a consequence, similarly to [130], in a scenario where the exporter expires records solely by means of timeouts, the number of *unhealthy* connections is slightly under-counted.

Results in Figure A.1 imply that timeouts of flow exporters can be set to any value without impacting our results. This is important because several exporters handle overloads by expiring flow records faster – *i.e.,* by reducing timeouts at run-time. The results of varying timeouts in Scenarios 2 and 3 are not shown, since the same flat lines in Figure A.1 are obtained in these scenarios.

Our extension to [130] improves the results when other expiration policies are applied in the flow exporter. Figure A.2(a) shows the $F$-measure for all scenarios when the idle and the active timeouts are fixed to 30 and 120 s, respectively. The low recall of the class *unhealthy* decreases the $F$-measure in Scenario 1, as previously explained. The improvement in Scenario 2 is caused by the proper checks of TCP flags in the flow exporter, which prevent different connections from being merged into a single flow record. Scenario 3 slightly improves the results for the same reason. The $F$-measure is close to 1 in Scenarios 2 and 3, meaning that both precision and recall are close to 1 in these cases – *i.e.,* the output of our method matches almost perfectly with the output of Bro.

Finally, Figure A.2(b) depicts the $F$-measure per class also for the original method presented in [130]. Only Scenario 3 is depicted, since the method in [130] produces similar results in other scenarios. Because we use more information to aggregate flow records, our method is more accurate when TCP flags are also considered for flow expiration (*i.e.,* Scenarios 2 and 3). Overall, results in this section show that, regardless of flow expiration policies and timeout parameters, our method is consistent when dealing with non-sampled flow records.

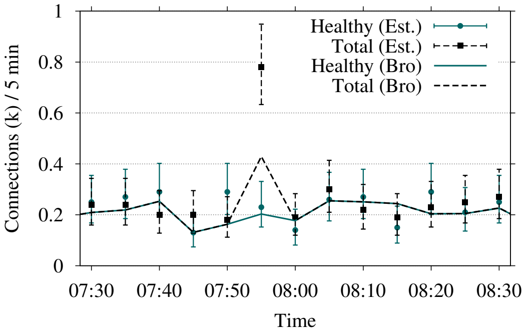


Figure A.3: An example of our estimations compared to the ground truth.

## A.3 Packet-Sampled Data

The validation with packet-sampled flows is performed against the same ground truth as in the previous section – *i.e.,* Bro output produced using non-sampled packet headers. Because sampling implies loss of information, it is not possible to output individually labeled connections as in the non-sampled case. Instead, confidence intervals of the total number of connections per class (*i.e., healthy* and *unhealthy*) are calculated. Similarly, expiration policies do not play the same fundamental role in the flow formation when sampling is applied, since only few packets per flow are observed. Hence, we validate whether our method calculates confidence intervals that include the original numbers counted from Bro output. For the results in this section, YAF converts our packet header dataset to packet-sampled flow records using $p = 0.1$. We first illustrate the application of our method in a simple example. After that, the method is applied to the complete dataset used in the previous section.

Figure A.3 shows the intervals calculated by our method and the number of connections reported by Bro using only the traffic going to Facebook in a limited time interval. Confidence intervals for $n$ and $n_h$ are calculated by Equation (3.1), with a significance level of 95 % and $r = 1$. Bro results show that the difference between $n$ and $n_h$ is very small most of the time. Around 7:55 AM, some users are not able to access Facebook, resulting in an increase in the number of connections. The confidence intervals always overlap, except on the slot at 7:55 AM. Hence, the service is correctly identified to be *unhealthy* at that time.

However, it is also clear that the estimation $\hat{n}$ strongly diverges from the real value $n$ when there are *unhealthy* connections. This happens because retransmissions of `SYN` packets violate our assumption of a single `SYN` packet per traffic direction on TCP flows, biasing the estimators. For the period in Figure A.3,

Table A.3: Connections with more than one SYN packet (%).

| Type | Frequency | Originator | Responder |
|---|---:|---:|---:|
| Ongoing | 1.4 | 19.9 | 51.9 |
| Closing | 0.7 | 2.4 | 2.3 |
| Aborted | 26.6 | 1.6 | 2.1 |
| Complete | 57.6 | 0.8 | 1.1 |
| Unhealthy | 13.7 | 46.9 | 0.2 |
| Total | 100.0 | 7.6 | 1.9 |

Bro reports 3,087 *healthy* connections out of 3,314 in total. In the same period, $\hat{n}_h = 3,450$ and $\hat{n} = 3,930$ under sampling.

When the method is applied to packet-sampled flows of the complete dataset, confidence intervals include the numbers reported by Bro in around 93 % of the intervals for the number of healthy connection $n_h$. On the other hand, the percentage is as low as 61 % for the total number of connections $n$. This percentage clearly diverges from the selected significance level (*i.e.,* 95 %). Retransmissions of SYN packets, in particular from originators, bias the intervals. Table A.3 explains that by reporting the percentage of flows with SYN retransmissions in the complete dataset. In particular, 46.9 % of the *unhealthy* connections include at least one SYN retransmission from originators. Therefore, the probability of sampling at least one SYN packet from these connections is higher.

For detecting availability problems, overestimating the *unhealthy* traffic is not harmful: it causes the *health index* to decrease faster in case of problems, thus helping in the detection. For illustration, however, next section presents a possible method to reduce the bias in the estimation.

### A.3.1 Handling SYN Retransmissions

Service failures can cause retransmissions of SYN packets, violating the assumptions made in Chapter 3 (see Section 3.1.2). Table A.3 shows that, not surprisingly, most retransmissions are from connection originators. In the following, we provide an estimation of $n$ taking into account that $n = \sum_{i \geq 1} n_i$, where $n_i$ is the number of connections with exactly $i$ SYN packets from originators.

For a given time interval, let $x_i$, $i \geq 1$, be the number of records with $i$ SYN packets. It should be noted that most NetFlow exporters do not report the number of observed SYN packets per record, although IPFIX exporters can use the tcpSynTotalCount information element to report that. We refer to [41] for a discussion on how $x_1, x_2, \ldots$ can be estimated directly from NetFlow records.

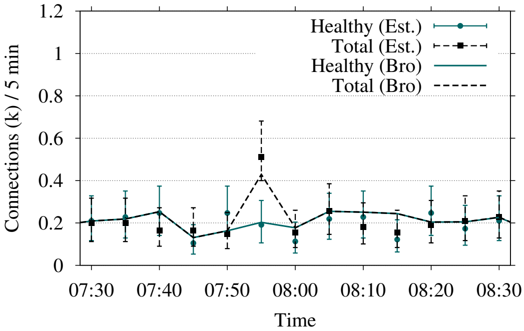


Figure A.4: An example of an estimation considering `SYN` retransmissions.

Given a TCP flow with $s$ `SYN` packets, the number $s'$ of `SYN` packets sampled for this flow follows the Binomial distribution $B(s'|s,p)$. In order to simplify the following calculations, we assume $n_i = 0$ for $i > 2$, *i.e.*, at most one `SYN` retransmission per TCP flow. Consequently, the probability of sampling the `SYN` packet of a flow with one `SYN` packet is $p$, the probability of sampling the two `SYN` packets from a flow with two `SYN` packets is $p^2$, and the probability of sampling one `SYN` packet from a flow with two `SYN` packets is $2p(1-p)$.

The probability of observing $x_2$ flow records with two `SYN` packets out of $n_2$ original flows with two `SYN` packets is $B(x_2|n_2,p^2)$. The probability of observing $x_{2,1}$ flow records with only one `SYN` packet out of the remaining $n_2 - x_2$ flows with two `SYN` packets is $B(x_{2,1}|n_2 - x_2, 2p(1-p))$. Finally, the probability of observing $x_{1,1}$ flow records with one `SYN` packet out of $n_1$ flows with one `SYN` packet is $B(x_{1,1}|n_1,p)$.

For small $p$, the above Binomial distributions can be approximated by Poisson distributions. Since $x_1 = x_{1,1}+x_{2,1}$, the conditional probability of observing $x_1$ flow records with one `SYN` packet is given by

$$f(x_1|n_1, n_2 - x_2, p) = \frac{\lambda_1^{x_1}}{x_1!}e^{-\lambda_1}, \tag{A.4}$$

with $\lambda_1 = n_1 p + 2(n_2 - x_2)p(1-p)$. The joint probability of observing $x_1, x_2$ is

$$f(x_1, x_2|n_1, n_2, p) = \frac{\lambda_1^{x_1}}{x_1!}\frac{\lambda_2^{x_2}}{x_2!}e^{-\lambda_1-\lambda_2}, \tag{A.5}$$

with $\lambda_2 = n_2 p^2$ and $\lambda_1$ as in Equation (A.4). Similarly to Equation (3.1) in Chapter 3, the posterior distribution of $n_1$ and $n_2$ can be estimated by

$$f(n_1, n_2|x_1, x_2, p) \propto f_0^{'}(n_1)f_0^{''}(n_2)\lambda_1^{x_1}\lambda_2^{x_2}e^{-\lambda_1-\lambda_2}, \tag{A.6}$$

where $f_0^{'}$ and $f_0^{''}$ are prior distributions of $n_1$ and $n_2$, respectively. Given the posterior distribution of $n_1$ and $n_2$, the posterior distribution of $n = n_1 + n_2$ can be computed numerically, and its mode $\hat{n}$ gives a maximum likelihood estimation of $n$. Confidence intervals are calculated by summing up the probabilities of neighbor values of the mode until the desired significance level is reached.

Figure A.4 illustrates the improvement when SYN retransmissions are taken into account in the estimations. In comparison to the results in Figure A.3, it can be seen that the estimation is more accurate when there are unhealthy connections. The estimation $\hat{n}$ considering the complete dataset used in the previous section includes the real value $n$ reported by Bro in 81 % of the intervals, instead of 61 %, in this case. Note that the approach in this section estimates $\hat{n}$ only from a single observation, in contrast to Equation (3.1) in Chapter 3, which considers a window of $r$ observations. Moreover, since we assumed that $n_i = 0$ for $i > 2$, some bias remains because this assumption might also be violated. Extending the estimator to other cases is out of the scope of this thesis.

APPENDIX B

# Dropbox Storage Traffic in Details

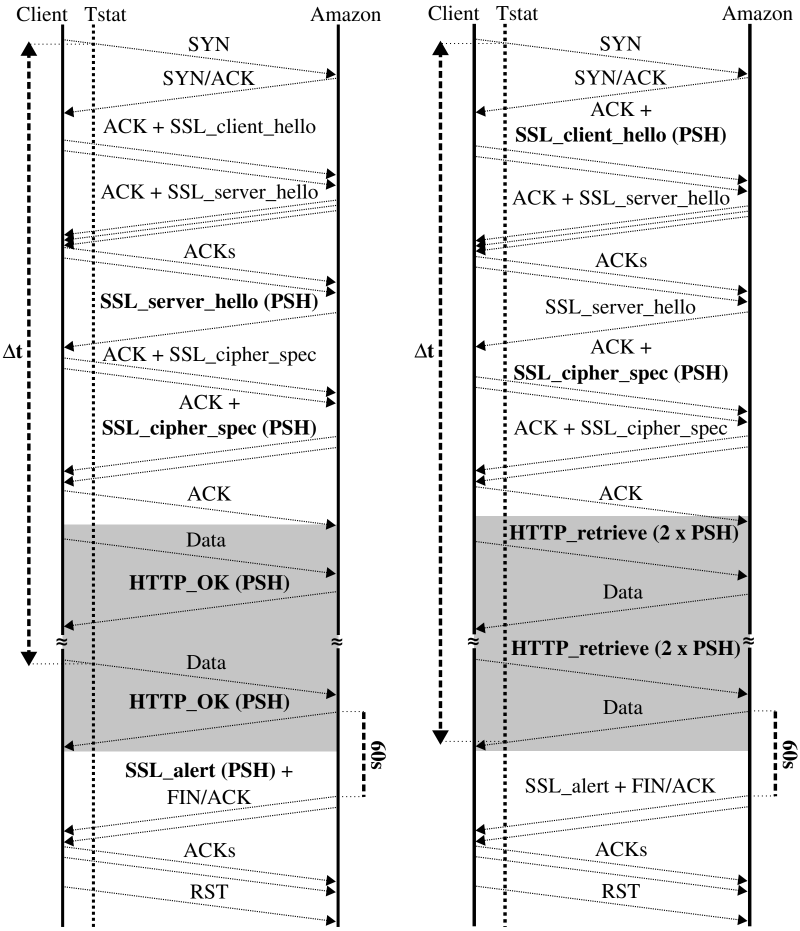
This appendix presents more details about the Dropbox client and its protocols. The information in the following helps to interpret flow measurements and calculate performance metrics that are meaningful to monitor the service at the application layer, as discussed in Chapter 4.

## B.1 Typical Storage Flows

Figure B.1 shows typical storage flows observed in our testbed (see Chapter 4). All packets exchanged during initial and final TCP and TLS/SSL handshakes are depicted. Initial and final handshakes are particularly constant in this case, because of the single client generating the traffic (*i.e.,* Dropbox). This constant behavior is key to our methodology, and allows us to exploit information visible at the transport layer, such as TCP flags, to derive performance metrics meaningful at the application layer. The data transfer phases (in gray) are shortened for the sake of clarity. Key elements to our methodology, such as TCP segments with `PSH` flag set and flow durations, are highlighted. To validate that these models are valid for real clients, Tstat in *Campus 1* has been set to record statistics about the first 10 messages delimited by TCP segments with `PSH` flag set. In the following, more details of our methodology and the results of this validation are presented.

## B.2 Tagging Storage Flows

Storage flows are first filtered using FQDNs and TLS/SSL certificate names. After that, they are classified based on the number of bytes sent by each endpoint of the TCP connection. The method has been built based on the assumption that a storage flow is used either for storing chunks or for retrieving chunks, but never for both. This assumption is supported by two facts: (i) when both operations happen in parallel, Dropbox uses separate connections to speed up



(a) Store (b) Retrieve

Figure B.1: Typical flows in storage operations.

synchronization; (ii) idle storage connections are kept open waiting for new commands only for a short time interval (*i.e.,* 60 s).

Our assumption could be possibly violated during this idle interval. In practice, however, this seems to be hardly the case. Figure B.2 illustrates that by plotting the number of bytes in storage flows in *Campus 1*. Flows are concentrated near the axes, as expected under our assumption.

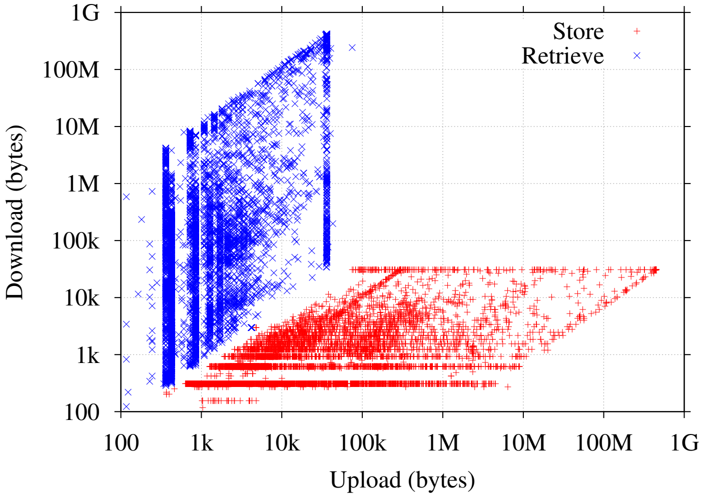


Figure B.2: Bytes exchanged in storage flows in *Campus 1*. Note the logarithmic scales.

Flows in Figure B.2 are already divided into two groups. The separation of flows into the groups can be defined by noting the gap between the regions in the figure, which is an outcome of typical overheads for each operation. The typical overhead of both *store* and *retrieve* operations could be documented based on the extra information collected in *Campus 1*. Note that TLS/SSL overheads are subtracted from each point in the figure for improving visualization.

Finally, we quantify the possible error caused by violations of our assumptions. In all vantage points, flows tagged as *store* download less than 1 % of the total storage volume. Since this includes protocol overhead (*e.g.,* HTTP_OK messages in Figure B.1), mixed flows marked as *store* might have only a negligible impact in our results. A similar reasoning is valid for *retrieve* flows.

## B.3 Number of Chunks

The number of chunks transported in a storage flow ($c$) is estimated by counting TCP segments with `PSH` flag set ($s$) in the reverse direction of the transfer, as indicated in Figure B.1. For *retrieve* flows, $c = \frac{s-2}{2}$. For *store* flows $c = s - 3$ or $c = s - 2$, depending on whether the connection is passively closed by the server or not. This can be inferred by the time difference between the last packet with payload from the client and the last one from the server: when the server closes

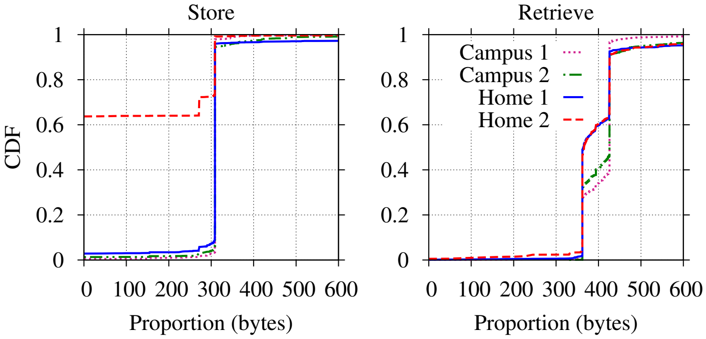


Figure B.3: Payload in the reverse direction of storage operations per estimated number of chunks.

an idle connection, the difference is expected to be around 1 min (otherwise, only a few seconds). Tstat already records the timestamps of such packets by default.

We validate this relation by dividing the amount of payload (without typical TLS/SSL handshakes) in the reverse direction of a transfer by $c$. This proportion has to be equal to the overhead needed per storage operation. Figure B.3 shows that for the vast majority of *store* flows the proportion is about 309 bytes per chunk, as expected given the extra data captured in *Campus 1* (see previous section). In *Home 2*, the apparently misbehaving client described in Chapter 4 biases the distribution: most flows from this client lack acknowledgment messages. Most *retrieve* flows have a proportion between 362 and 426 bytes per chunk, which are typical sizes of the HTTP request in this command. The exceptions (3 % – 8 %) might be caused by packet loss in our probes as well as by the flows that both stored and retrieved chunks. Our method underestimates the number of chunks in those cases.

## B.4 Duration

Figure B.1 shows the transfer duration ($\Delta t$ – see also Section 4.4.4) used when computing the throughput of a storage flow. Since initial TCP and TLS/SSL handshakes affect users' perception of throughput, the first `SYN` packet is taken as the begin of the transfer. Termination handshakes, on the other hand, are ignored. In *store* flows, the last packet with payload sent by the client is considered the end of the transfer. In *retrieve* flows, the last packet with payload is normally a server alert about the TLS/SSL termination. We compensate for that by subtracting 60 s from the duration of *retrieve* flows whenever the difference between the last packet with payload from the server and the one from the client is above 60 s.

Because of our monitoring topology, $\Delta t$ does not include the trip time between clients and our probes. $\Delta t$ is, therefore, slightly underestimated. Figure B.1 also shows that 4 or 5 RTTs are needed before the client starts to send or to receive data. In some cases, this already accounts for about 500 ms in the flow duration. Note that the initial TCP congestion window in place at servers forces a pause of 1 RTT during the TLS/SSL handshake. This parameter has been tuned after the release of Dropbox 1.4.0, thus reducing the overhead.

# Bibliography

[1] Agrawal, N., Bolosky, W. J., Douceur, J. R., and Lorch, J. R. 2007. A Five-Year Study of File-System Metadata. *ACM Transactions on Storage 3,* 3.

[2] Al-Fares, M., Elmeleegy, K., Reed, B., and Gashinsky, I. 2011. Overclocking the Yahoo! CDN for Faster Web Page Loads. In *Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement.* IMC'11. 569–584.

[3] Amazon. Cloud Drive. `http://www.amazon.com/gp/feature.html?docId=1000828861`. Online. Accessed May 2013.

[4] Amazon. Web Services. `http://aws.amazon.com/`. Online. Accessed Jul 2013.

[5] Anderson, S., Niccolini, S., and Hogrefe, D. 2009. SIPFIX: A Scheme For Distributed SIP Monitoring. In *Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management.* IM'09. 382–389.

[6] Andrejevic, M. 2007. *iSpy: Surveillance and Power in the Interactive Era* 1 Ed. University Press of Kansas, Lawrence, KS, USA.

[7] Anti, M. 2012. Behind the Great Firewall of China. TEDTalk `http://www.ted.com/talks/michael_anti_behind_the_great_firewall_of_china.html`. Online. Accessed May 2013.

[8] Arlitt, M. and Williamson, C. 2005. An Analysis of TCP Reset Behaviour on the Internet. *ACM SIGCOMM Computer Communication Review 35,* 1, 37–44.

[9] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. 2010. A View of Cloud Computing. *Communications of the ACM 53*, 50–58.

[10] Arthur, C. 2010. Google's ChromeOS Means Losing Control of Data, Warns GNU Founder Richard Stallman. `http://www.guardian.co.uk/technology/blog/2010/dec/14/chrome-os-richard-stallman-warning`. Online. Accessed Jul 2013.

[11] Bentham, J. 1791. *Panopticon; or, the Inspection-House.* Vol. 2. London, UK.

[12] Bergen, A., Coady, Y., and McGeer, R. 2011. Client Bandwidth: The Forgotten Metric of Online Storage Providers. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing.* PacRim'2011. 543–548.

[13] Bermudez, I., Mellia, M., Munafò, M. M., Keralapura, R., and Nucci, A. 2012. DNS to the Rescue: Discerning Content and Services in a Tangled Web. In *Proceedings of the 12th ACM Internet Measurement Conference*. IMC'12. 413–426.

[14] Bermudez, I. N., Traverso, S., Mellia, M., and Munafò, M. M. 2013. Exploring the Cloud from Passive Measurements: The Amazon AWS Case. In *Proceedings of the 32nd Annual IEEE International Conference on Computer Communications*. INFOCOM'13.

[15] Berthier, R., Cukier, M., Hiltunen, M., Kormann, D., Vesonder, G., and Sheleheda, D. 2010. Nfsight: NetFlow-Based Network Awareness Tool. In *Proceedings of the 24th International Conference on Large Installation System Administration*. LISA'10. 1–8.

[16] Bracewell, R. N. 1986. *The Fourier Transform and Its Applications* 2 Ed. McGraw-Hill, New York, NY, USA.

[17] Brignall, T. 2002. The New Panopticon: The Internet Viewed as a Structure of Social Control. *Theory & Science 3,* 1.

[18] Brownlee, N. 1999. Traffic Flow Measurement: Meter MIB. RFC 2720 (Standards Track).

[19] Brownlee, N. 2011. Flow-Based Measurement: IPFIX Development and Deployment. *IEICE Transactions on Communications 94,* 8, 2190–2198.

[20] Brownlee, N., Mills, C., and Ruth, G. 1999. Traffic Flow Measurement: Architecture. RFC 2722 (Informational).

[21] Callado, A., Kamienski, C., Szabó, G., Gero, B. P., Kelner, J., Fernandes, S., and Sadok, D. 2009. A Survey on Internet Traffic Identification. *IEEE Communications Surveys & Tutorials 11,* 3, 37–52.

[22] Čeleda, P., Kováčik, M., Koníř, T., Krmíček, V., Špringl, P., and Žádník, M. 2007. FlowMon Probe. Tech. rep., CESNET.

[23] Cha, M., Kwak, H., Rodriguez, P., Ahn, Y.-Y., and Moon, S. 2007. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC'07. 1–14.

[24] Cisco Systems. 2006. IOS Flexible NetFlow Overview. `http://www.cisco.com/en/US/docs/ios/fnetflow/configuration/guide/fnetflow_overview.html`. Online. Accessed Jun 2013.

[25] Cisco Systems. 2009. Catalyst 6500 Series Switch Cisco IOS Software Configuration Guide. `http://www.cisco.com/en/US/docs/switches/lan/catalyst6500/ios/12.2SXF/native/configuration/guide/122sxscg.pdf`. Online. Accessed Dec 2012.

[26] Claffy, K. C., Braun, H.-W., and Polyzos, G. C. 1995. A Parameterizable Methodology for Internet Traffic Flow Profiling. *IEEE Journal on Selected Areas in Communications 13,* 8, 1481–1494.

[27] Claise, B. 2004. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational).

[28] Claise, B. 2008. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Standards Track).

[29] Claise, B., Aitken, P., and Ben-Dvora, N. 2012. Cisco Systems Export of Application Information in IP Flow Information Export (IPFIX). RFC 6759 (Informational).

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

[31] Clarke, R. 2012. How Reliable is Cloudsourcing? A Review of Articles in the Technical Media 2005-11. *Computer Law & Security Review 28,* 1, 90–95.

[32] CloudSleuth. Global Provider View. `https://cloudsleuth.net/global-provider-view`. Online. Accessed Jul 2013.

[33] Compuware. APM Synthetic Monitoring. `http://www.compuware.com/en_us/application-performance-management/products/user-experience-management/synthetic-web-and-mobile/overview.html`. Online. Accessed Jul 2013.

[34] Cunha, I., Silveira, F., Oliveira, R., Teixeira, R., and Diot, C. 2009. Uncovering Artifacts of Flow Measurement Tools. In *Proceedings of the 10th International Conference on Passive and Active Network Measurement.* PAM'09. 187–196.

[35] de O. Schmidt, R., Sperotto, A., Sadre, R., and Pras, A. 2012. Towards Bandwidth Estimation using Flow-Level Measurements. In *Proceedings of the 6th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security.* AIMS'12. 127–138.

[36] Deri, L. 2003. nProbe: An Open Source NetFlow Probe for Gigabit Networks. In *Proceedings of the Terena.* TNC'03.

[37] Deri, L. 2006. Open Source VoIP Traffic Monitoring. In *Proceedings of the 5th System Administration and Network Engineering Conference.* SANE'06.

[38] Deri, L., Trombacchi, L. L., Martinelli, M., and Vannozzi, D. 2012. A Distributed DNS Traffic Monitoring System. In *Proceedings of the 8th International Wireless Communications and Mobile Computing Conference.* IWCMC,12. 30–35.

[39] Drago, I., Barbosa, R. R., Sadre, R., Pras, A., and Schönwälder, J. 2011a. Report of the Second Workshop on the Usage of NetFlow/IPFIX in Network Management. *Journal of Network and Systems Management 19,* 2, 298–304.

[40] Drago, I., Bocchi, E., Mellia, M., Slatman, H., and Pras, A. 2013a. Benchmarking Personal Cloud Storage. In *Proceedings of the 13th ACM Internet Measurement Conference.* IMC'13.

[41] Drago, I., Hofstede, R., Sadre, R., Sperotto, A., and Pras, A. 2013b. Measuring Cloud Service Health using NetFlow/IPFIX: The WikiLeaks Case. *Journal of Network and Systems Management.* Online First Articles. `http://dx.doi.org/10.1007/s10922-013-9278-0`. Accessed Jun 2013.

[42] Drago, I., Mellia, M., Munafò, M. M., Sperotto, A., Sadre, R., and Pras, A. 2012. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the 12th ACM Internet Measurement Conference.* IMC'12. 481–494.

[43] Drago, I. and Pras, A. 2010. Scalable Service Performance Monitoring. In *Proceedings of the 4th International Conference on Autonomous Infrastructure, Management and Security.* AIMS'10. 175–178.

[44] Drago, I., Sadre, R., and Pras, A. 2011b. Report of the Third Workshop on the Usage of NetFlow/IPFIX in Network Management. *Journal of Network and Systems Management 19,* 4, 529–535.

[45] Draper, N. and Guttman, I. 1971. Bayesian Estimation of the Binomial Parameter. *Technometrics 13,* 3, 667–673.

[46] Dropbox. Dropbox Release Notes. `https://www.dropbox.com/release_notes/`. Online. Accessed May 2013.

[47] Dropbox. DropboxOps. `http://twitter.com/DropboxOps`. Online. Accessed Jun 2013.

[48] Dropbox. What Is Dropbox? `https://www.dropbox.com/news/company-info/`. Online. Accessed May 2013.

[49] Duffield, N. 2004. Sampling for Passive Internet Measurement: A Review. *Statistical Science 19,* 3, 472–498.

[50] Duffield, N. and Lund, C. 2003. Predicting Resource Usage and Estimation Accuracy in an IP Flow Measurement Collection Infrastructure. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement.* IMC'03. 179–191.

[51] Duffield, N., Lund, C., and Thorup, M. 2002. Properties and Prediction of Flow Statistics from Sampled Packet Streams. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement.* IMW'02. 159–171.

[52] DUFFIELD, N., LUND, C., AND THORUP, M. 2005. Estimating Flow Distributions from Sampled Flow Statistics. *IEEE/ACM Transactions on Networking 13,* 5, 933–946.

[53] DUKKIPATI, N., REFICE, T., CHENG, Y., CHU, J., HERBERT, T., AGARWAL, A., JAIN, A., AND SUTIN, N. 2010. An Argument for Increasing TCP's Initial Congestion Window. *ACM SIGCOMM Computer Communication Review 40,* 3, 26–33.

[54] DURKEE, D. 2010. Why Cloud Computing Will Never Be Free. *ACM Queue 8,* 4, 20–29.

[55] ERIKSSON, B. AND CROVELLA, M. 2013. Understanding Geolocation Accuracy using Network Geometry. In *Proceedings of the 32nd Annual IEEE International Conference on Computer Communications.* INFOCOM'13.

[56] FAWCETT, T. 2006. An Introduction to ROC Analysis. *Pattern Recognition Letters 27,* 8, 861–874.

[57] FILIPPI, P. D. AND MCCARTHY, S. 2012. Cloud Computing: Centralization and Data Sovereignty. *European Journal of Law and Technology 3,* 2.

[58] FINAMORE, A., GEHLEN, V., MELLIA, M., MUNAFÒ, M. M., AND NICOLINI, S. 2012. The Need for an Intelligent Measurement Plane: The Example of Time-Variant CDN Policies. In *Proceedings of 15th International Telecommunications Network Strategy and Planning Symposium.* NETWORKS'12. 1–6.

[59] FINAMORE, A., MELLIA, M., MEO, M., MUNAFÒ, M. M., AND ROSSI, D. 2011a. Experiences of Internet Traffic Monitoring with Tstat. *IEEE Network 25,* 3, 8–14.

[60] FINAMORE, A., MELLIA, M., MUNAFÒ, M. M., TORRES, R., AND RAO, S. G. 2011b. YouTube Everywhere: Impact of Device and Infrastructure Synergies on User Experience. In *Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement.* IMC'11. 345–360.

[61] FOLLETT, J. H. 2006. Cisco: Catalyst 6500 The Most Successful Switch Ever. `http://www.cisco.com/en/US/docs/switches/lan/catalyst6500/ios/12.2SXF/native/configuration/guide/122sxscg.pdf`. Online. Accessed Jun 2013.

[62] FOUCAULT, M. 1995. *Discipline & Punish: The Birth of the Prison* 2 Ed. Vintage, New York, NY, USA.

[63] FULLMER, M. AND ROMIG, S. 2000. The OSU Flow-Tools Package and CISCO NetFlow Logs. In *Proceedings of the 14th USENIX Conference on System Administration.* LISA'00. 291–304.

[64] García-Dorado, J. L., Finamore, A., Mellia, M., Meo, M., and Munafò, M. M. 2012. Characterization of ISP Traffic: Trends, User Habits, and Access Technology Impact. *IEEE Transactions on Network and Service Management 9,* 2, 142–155.

[65] García-Dorado, J. L., Mata, F., Ramos, J., del Río, P. M. S., Moreno, V., and Aracil, J. 2013. High-Performance Network Traffic Processing Systems using Commodity Hardware. In *Data Traffic Monitoring and Analysis*. LNCS Series, vol. 7754. 3–27.

[66] Gehlen, V., Finamore, A., Mellia, M., and Munafò, M. M. 2012. Uncovering the Big Players of the Web. In *Proceedings of the 4th International Conference on Traffic Monitoring and Analysis*. TMA'12. 15–28.

[67] Gjoka, M., Sirivianos, M., Markopoulou, A., and Yang, X. 2008. Poking Facebook: Characterization of OSN Applications. In *Proceedings of the 1st Workshop on Online Social Networks*. WOSN'08. 31–36.

[68] Glatz, E. and Dimitropoulos, X. 2012. Classifying Internet One-way Traffic. In *Proceedings of the 12th ACM Internet Measurement Conference*. IMC'12. 37–50.

[69] Goodin, D. 2012. Yes, Microsoft Azure Was Downed By Leap-Year Bug. `http://www.wired.com/wiredenterprise/2012/03/azure-leap-year-bug/`. Online. Accessed Jul 2013.

[70] Google. Apps Status Dashboard. `http://www.google.com/appsstatus`. Online. Accessed Jun 2013.

[71] Google. Docs. `https://docs.google.com/`. Online. Accessed Jul 2013.

[72] Google. Drive. `https://tools.google.com/dlpage/drive/`. Online. Accessed May 2013.

[73] Google. Network Introduction. `https://peering.google.com/about/delivery_ecosystem.html`. Online. Accessed May 2013.

[74] Google. Trends. `http://www.google.com/trends/`. Online. Accessed May 2013.

[75] Greenwald, G. and MacAskill, E. 2013. NSA Prism Program Taps In to User Data of Apple, Google and Others. `http://www.guardian.co.uk/world/2013/jun/06/us-tech-giants-nsa-data`. Online. Accessed Jul 2013.

[76] Gu, Y., Breslau, L., Duffield, N., and Sen, S. 2009. On Passive One-Way Loss Measurements using Sampled Flow Statistics. In *Proceedings of the 28th Annual IEEE International Conference on Computer Communications*. INFOCOM'09. 2946–2950.

[77] Haag, P. Plugin Writers Guide. `http://nfsen.sourceforge.net/PluginGuide/plugin-guide.html`. Online. Accessed Jun 2013.

[78] Haag, P. 2005. Watch Your Flows with NfSen and NFDUMP. 50th RIPE Meeting `http://meetings.ripe.net/ripe-50/presentations/`. Online. Accessed May 2013.

[79] Habib, S. M., Ries, S., and Muhlhauser, M. 2010. Cloud Computing Landscape and Research Challenges regarding Trust and Reputation. In *Proceedings of the Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing.* UIC-ATC'10. 410–415.

[80] Hajjat, M., Sun, X., Sung, Y.-W. E., Maltz, D., Rao, S., Sripanidkulchai, K., and Tawarmalani, M. 2010. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. *ACM SIGCOMM Computer Communication Review 40,* 4, 243–254.

[81] Halevi, S., Harnik, D., Pinkas, B., and Shulman-Peleg, A. 2011. Proofs of Ownership in Remote Storage Systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security.* CCS'11. 491–500.

[82] Harnik, D., Pinkas, B., and Shulman-Peleg, A. 2010. Side Channels in Cloud Services: Deduplication in Cloud Storage. *IEEE Security and Privacy 8,* 6, 40–47.

[83] Hätönen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., and Kojo, M. 2010. An Experimental Study of Home Gateway Characteristics. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement.* IMC'10. 260–266.

[84] Höfer, C. and Karagiannis, G. 2011. Cloud Computing Services: Taxonomy and Comparison. *Journal of Internet Services and Applications 2*, 81–94.

[85] Hofstede, R., Drago, I., Sperotto, A., Sadre, R., and Pras, A. 2013. Measurement Artifacts in NetFlow Data. In *Proceedings of the 14th International Conference on Passive and Active Measurement.* PAM'13. 1–10.

[86] Hofstede, R. and Fioreze, T. 2009. SURFmap: A Network Monitoring Tool based on the Google Maps API. In *Proceedings of the 11th IFIP/IEEE International Conference on Symposium on Integrated Network Management.* IM'09. 676–690.

[87] Hu, W., Yang, T., and Matthews, J. N. 2010. The Good, the Bad and the Ugly of Consumer Cloud Storage. *ACM SIGOPS Operating Systems Review 44,* 3, 110–115.

[88] IANA. 2007. IP Flow Information Export (IPFIX) Entities. `http://www.iana.org/assignments/ipfix/ipfix.xml`. Online. Accessed Jun 2013.

[89] Inacio, C. M. and Trammell, B. 2010. YAF: Yet Another Flowmeter. In *Proceedings of the 24th International Conference on Large Installation System Administration.* LISA'10. 1–16.

[90] John, W., Tafvelin, S., and Olovsson, T. 2010. Passive Internet Measurement: Overview and Guidelines based on Experiences. *Computer Communications 33,* 5, 533–550.

[91] Kannan, J., Jung, J., Paxson, V., and Koksal, C. E. 2006. Semi-Automated Discovery of Application Session Structure. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement.* IMC'06. 119–132.

[92] Kögel, J. 2011. One-way Delay Measurement based on Flow Data: Quantification and Compensation of Errors by Exporter Profiling. In *Proceedings of the International Conference on Information Networking.* ICOIN'11. 25–30.

[93] Kossmann, D., Kraska, T., and Loesing, S. 2010. An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* SIGMOD'10. 579–590.

[94] Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., and Jahanian, F. 2010. Internet Inter-Domain Traffic. In *Proceedings of the ACM SIGCOMM 2010 Conference.* SIGCOMM'10. 75–86.

[95] LaCie. Wuala. `http://www.wuala.com/`. Online. Accessed May 2013.

[96] Lampert, R. T., Sommer, C., Munz, G., and Dressler, F. 2006. Vermont – A Versatile Monitoring Toolkit for IPFIX and PSAMP. In *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation.* MonAM'06.

[97] Leinen, S. 2004. Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX). RFC 3955 (Informational).

[98] Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. 2009. What's Inside the Cloud? An Architectural Map of the Cloud Landscape. In *Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing.* CLOUD'09. 23–31.

[99] Lenk, A., Menzel, M., Lipsky, J., Tai, S., and Offermann, P. 2011. What Are You Paying For? Performance Benchmarking for Infrastructure-as-a-Service Offerings. In *Proceedings of the 4th IEEE International Conference on Cloud Computing.* CLOUD'11. 484–491.

[100] Lewis, D. D. and Gale, W. A. 1994. A Sequential Algorithm for Training Text Classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* SIGIR'94. 3–12.

[101] Li, A., Yang, X., Kandula, S., and Zhang, M. 2010. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement.* IMC'10. 1–14.

[102] Li, B., Springer, J., Bebis, G., and Gunes, M. H. 2013. A Survey of Network Flow Applications. *Journal of Network and Computer Applications 36,* 2, 567–581.

[103] Limmer, T. and Dressler, F. 2009. Flow-Based TCP Connection Analysis. In *Proceedings of the 2nd IEEE International Workshop on Information and Data Assurance*. WIDA'09. 376–383.

[104] Mager, T., Biersack, E., and Michiardi, P. 2012. A Measurement Study of the Wuala On-line Storage Service. In *Proceedings of the IEEE 12th International Conference on Peer-to-Peer Computing*. P2P'12. 237–248.

[105] Mansfield-Devine, S. 2011. Anonymous: Serious Threat or Mere Annoyance? *Network Security 2011,* 1, 4–10.

[106] MaxMind. GeoIP Organization. `http://www.maxmind.com/en/organization`. Online. Accessed May 2013.

[107] Mellia, M., Meo, M., Muscariello, L., and Rossi, D. 2008. Passive Analysis of TCP Anomalies. *Computer Networks 52,* 14, 2663–2676.

[108] Meng, S., Iyengar, A. K., Rouvellou, I. M., Liu, L., Lee, K., Palanisamy, B., and Tang, Y. 2012. Reliable State Monitoring in Cloud Datacenters. In *Proceedings of the 5th IEEE International Conference on Cloud Computing*. CLOUD'12. 951–958.

[109] Meng, S. and Liu, L. 2012. Enhanced Monitoring-as-a-Service for Effective Cloud Management. *IEEE Transactions on Computers PP*.

[110] Microsoft. SkyDrive. `https://skydrive.live.com/`. Online. Accessed May 2013.

[111] Mislove, A., Marcon, M., Gummadi, K. P., Druschel, P., and Bhattacharjee, B. 2007. Measurement and Analysis of Online Social Networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC'07. 29–42.

[112] Mitseva, A. 2012. On-line Monitoring of Cloud Providers with NfSen. Bachelor thesis, Technical University of Sofia.

[113] Mulazzani, M., Schrittwieser, S., Leithner, M., Huber, M., and Weippl, E. 2011. Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space. In *Proceedings of the 20th USENIX Conference on Security*. SEC'11.

[114] Muthitacharoen, A., Chen, B., and Mazières, D. 2001. A Low-Bandwidth Network File System. *ACM SIGOPS Operating Systems Review 35,* 5, 174–187.

[115] Patterson, M. 2009. NetFlow v9 vs. NetFlow v5: What Are the Differences? http://www.plixer.com/blog/netflow/netflow-v9-vs-netflow-v5/. Online. Accessed Jun 2013.

[116] Paxson, V. 1999. Bro: A System for Detecting Network Intruders in Real-time. *Computer Networks 31,* 23-24, 2435–2463.

[117] Paxson, V. 2004. Strategies for Sound Internet Measurement. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement.* IMC'04. 263–271.

[118] Poese, I., Uhlig, S., Kaafar, M. A., Donnet, B., and Gueye, B. 2011. IP Geolocation Databases: Unreliable? *ACM SIGCOMM Computer Communication Review 41,* 2, 53–56.

[119] Portugal Telecom. SmartCloudPT. https://www.smartcloudpt.pt/Pages/FAQ/. Online. Accessed Jul 2013.

[120] Pras, A., Sperotto, A., Moura, G. C., Drago, I., Barbosa, R. R., Sadre, R., de O. Schmidt, R., and Hofstede, R. 2010. Attacks by "Anonymous" WikiLeaks Proponents not Anonymous. Tech. Rep. TR-CTIT-10-41, CTIT, University of Twente.

[121] Ptáček, L. 2011. Analysis and Detection of Skype Network Traffic. Master thesis, Masaryk University.

[122] Quittek, J., Zseby, T., Claise, B., and Zander, S. 2004. Requirements for IP Flow Information Export (IPFIX). RFC 3917 (Informational).

[123] Rimal, B. P., Choi, E., and Lumb, I. 2009. A Taxonomy and Survey of Cloud Computing Systems. In *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC.* NCM'09. 44–51.

[124] Rousskov, A. and Tsantilas, C. Squid-in-the-Middle SSL Bump. http://wiki.squid-cache.org/Features/SslBump/. Online. Accessed May 2013.

[125] Rullgard, M. Magic Number Recognition Library. http://sourceforge.net/projects/libmagic/. Online. Accessed Jun 2013.

[126] Sadasivan, G., Brownlee, N., Claise, B., and Quittek, J. 2009. Architecture for IP Flow Information Export. RFC 5470 (Informational).

[127] Sadre, R. and Haverkort, B. R. 2008. Fitting Heavy-Tailed HTTP Traces with the New Stratified EM-Algorithm. In *Proceedings of the 4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks.* IT-NEWS. 254–261.

[128] Schatzmann, D., Leinen, S., Kögel, J., and Mühlbauer, W. 2011. FACT: Flow-Based Approach for Connectivity Tracking. In *Proceedings of the 12th International Conference on Passive and Active Network Measurement.* PAM'11. 214–223.

[129] Schönwälder, J., Fouquet, M., Rodosek, G. D., and Hochstatter, I. C. 2009. Future Internet = Content + Services + Management. *IEEE Communications Magazine 47,* 7, 27–33.

[130] Sommer, R. and Feldmann, A. 2002. NetFlow: Information Loss or Win? In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement.* IMW'02. 173–174.

[131] Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., and Stiller, B. 2010. An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys & Tutorials 12,* 3, 343–356.

[132] Steinberger, J., Schehlmann, L., Abt, S., and Baier, H. 2013. Anomaly Detection and Mitigation at Internet Scale: a Survey. In *Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security.* AIMS'13. 49–60.

[133] Tang, V. K. and Sindler, R. B. 1987. Confidence Interval for Parameter n in a Binomial Distribution. Tech. Rep. CRM 86-265, Center for Naval Analyses.

[134] Tang, V. K., Sindler, R. B., and Shirven, R. M. 1987. Bayesian Estimation of n in a Binomial Distribution. Tech. Rep. CRM 87-185, Center for Naval Analyses.

[135] Thousandeyes. Performance Management for the Cloud Era. `http://www.thousandeyes.com/`. Online. Accessed Jul 2013.

[136] Torres, R., Finamore, A., Kim, J. R., Mellia, M., Munafò, M. M., and Rao, S. 2011. Dissecting Video Server Selection Strategies in the YouTube CDN. In *Proceedings of the 31st International Conference on Distributed Computing Systems.* ICDCS'11. 248–257.

[137] Trammell, B. and Boschi, E. 2008. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103 (Standards Track).

[138] Trammell, B. and Boschi, E. 2011. An Introduction to IP Flow Information Export (IPFIX). *IEEE Communications Magazine 49,* 4, 89–95.

[139] Trammell, B., Tellenbach, B., Schatzmann, D., and Burkhart, M. 2011. Peeling away Timing Error in NetFlow Data. In *Proceedings of the 12th International Conference on Passive and Active Measurement.* PAM'11. 194–203.

[140] Tridgell, A. 1999. Efficient Algorithms for Sorting and Synchronization. Ph.D. thesis, Australian National University. `http://www.samba.org/~tridge/phd_thesis.pdf`.

[141] Trifilò, A., Burschka, S., and Biersack, E. 2009. Traffic to Protocol Reverse Engineering. In *Proceedings of the 2nd IEEE International Conference on Computational Intelligence for Security and Defense Applications.* CISDA'09. 257–264.

[142] TWITTER. Status. `http://status.twitter.com`. Online. Accessed Jun 2013.

[143] VAN RIJSBERGEN, C. 1979. *Information Retrieval* 2 Ed. Butterworth, London, UK.

[144] VAQUERO, L. M., RODERO-MERINO, L., CACERES, J., AND LINDNER, M. 2008. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review 39,* 1, 50–55.

[145] VIJAYAN, J. 2013. U.S. Cloud Firms Face Backlash from NSA Spy Programs. `http://www.computerworlduk.com/news/security/3460473/us-cloud-firms-face-backlash-from-nsa-spy-programs/`. Online. Accessed Jul 2013.

[146] WALDBUSSER, S. 2004. Application Performance Measurement MIB. RFC 3729 (Standards Track).

[147] WANG, G. AND NG, T. E. 2010. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *Proceedings of the 29th Annual IEEE International Conference on Computer Communications*. INFOCOM'10. 1–9.

[148] WANG, H., SHEA, R., WANG, F., AND LIU, J. 2012. On the Impact of Virtualization on Dropbox-Like Cloud File Storage/Synchronization Services. In *Proceedings of the IEEE 20th International Workshop on Quality of Service*. IWQoS '12. 11:1–11:9.

[149] ZHANG, J. AND MOORE, A. 2007. Traffic Trace Artifacts due to Monitoring Via Port Mirroring. In *Proceedings of the 15th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*. E2EMON'07. 1–8.

[150] ZHANG, Q., CHENG, L., AND BOUTABA, R. 2010. Cloud Computing: State-of-the-Art and Research Challenges. *Journal of Internet Services and Applications 1*, 7–18.

[151] ZSEBY, T., BOSCHI, E., BROWNLEE, N., AND CLAISE, B. 2009a. IP Flow Information Export (IPFIX) Applicability. RFC 5472 (Informational).

[152] ZSEBY, T., MOLINA, M., DUFFIELD, N., NICCOLINI, S., AND RASPALL, F. 2009b. Sampling and Filtering Techniques for IP Packet Selection. RFC 5475 (Standards Track).

# Acronyms

**ADSL** Asymmetric Digital Subscriber Line

**Amazon EC2** Amazon Elastic Compute Cloud

**Amazon S3** Amazon Simple Storage Service

**API** Application Programming Interface

**AS** Autonomous System

**AWS** Amazon Web Service

**CDF** Cumulative Distribution Function

**CPU** Central Processing Unit

**DNS** Domain Name System

**DPI** Deep Packet Inspection

**FQDN** Fully Qualified Domain Name

**FTP** File Transfer Protocol

**FTTH** Fiber to the Home

**HTTP** Hypertext Transfer Protocol

**HTTPS** Hypertext Transfer Protocol Secure

**HuaaS** Human as a Service

**IaaS** Infrastructure as a Service

**IANA** Internet Assigned Numbers Authority

**IETF** Internet Engineering Task Force

**IP** Internet Protocol

**IPFIX** IP Flow Information Export

**ISP** Internet Service Provider

**LAN** Local Area Network

**MIB** Management Information Base

**MPLS** Multi-Protocol Label Switching

**NAT** Network Address Translation

**NSA** United States National Security Agency

**PaaS** Platform as a Service

**POP** Point of Presence

**PSAMP** Packet SAMPling

**RFC** Request for Comment

**RRD** Round-Robin Database

**RTFM** Real-time Traffic Flow Measurement

**RTT** Round Trip Time

**SaaS** Software as a Service

**SLA** Service Level Agreement

**SNMP** Simple Network Management Protocol

**SSL** Secure Sockets Layer

**TCP** Transmission Control Protocol

**TLS** Transport Layer Security

**UDP** User Datagram Protocol

**URL** Uniform Resource Locator

**UT** University of Twente

**VoIP** Voice over IP

# About the author

I was born in Marilândia, Espírito Santo, Brazil, on March 30th, 1980. I received my Master of Science (M.Sc.) degree in Computer Science in 2007 and my Bachelor of Science (B.Sc.) degree in Computer Engineering in 2004, both from the Federal University of Espírito Santo, Brazil. From 2009 until 2013, I was a Ph.D. student at the Design and Analysis of Communication Systems Group (DACS) of the University of Twente, under supervision of Prof.dr.ir. Aiko Pras and Prof.dr.ir Boudewijn R. Haverkort. During my Ph.D. I spent 3 months in an internship in the Politecnico di Torino, in Italy, where I had the pleasure to work with Prof. Maurizio M. Munafò and Prof. Marco Mellia. These are the papers I published during the time I was a Ph.D. student:

- Drago, I., Bocchi, E., Mellia, M., Slatman, H., and Pras, A. 2013. *Benchmarking Personal Cloud Storage.* In Proceedings of the 13th ACM Internet Measurement Conference. IMC'13.
- Hofstede, R., Drago, I., Sperotto, A., Sadre, R., and Pras, A. 2013. *Measurement Artifacts in NetFlow Data.* In Proceedings of the 14th International Conference on Passive and Active Measurement. PAM'13. pp. 1–10.
  **Best Paper Award of PAM 2013**.
- Drago, I., Hofstede, R., Sadre, R., Sperotto, A., and Pras, A. 2013. *Measuring Cloud Service Health using NetFlow/IPFIX: the WikiLeaks Case.* Journal of Network and Systems Management. Accepted for publication.
- Drago, I. and Vieira, A.B. and Silva, A.P.C. 2013. (in Portuguese) *Caracterização dos Arquivos Armazenados no Dropbox.* In: WP2P+. 31° Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. SBRC'13.
- Drago, I., Mellia, M., Munafò, M. M., Sperotto, A., Sadre, R., and Pras, A. 2012. *Inside Dropbox: Understanding Personal Cloud Storage Services.* In Proceedings of the 12th ACM Internet Measurement Conference. IMC'12. pp. 481–494.
  **IETF/IRTF Applied Networking Research Prize 2013**.
- Hofstede, R.J. and Drago, I. and Sperotto, A. and Pras, A. 2011 *Flow Monitoring Experiences at the Ethernet-Layer.* In: Proceedings of the 17th Workshop on Energy-Aware Communications, EUNICE'11. pp. 129–140.
- Hofstede, R.J. and Drago, I. and Moreira Moura, G.C. and Pras, A. 2011. *Carrier Ethernet OAM: An Overview and Comparison to IP OAM.* In: Proceedings

of the 5th International Conference on Autonomous Infrastructure, Management and Security, AIMS'11, pp. 112–123.

- Drago, I. and Sadre, R. and Pras, A. 2011. *Report of the Third Workshop on the Usage of NetFlow/IPFIX in Network Management.* Journal of network and systems management, 19 (4). pp. 529-535.
- Drago, I. and Barbosa, R.R.R. and Sadre, R. and Pras, A. and Schönwälder, J. 2011. *Report of the Second Workshop on the Usage of NetFlow/IPFIX in Network Management.* Journal of Network and Systems Management, 19 (2). pp. 298-304.
- Pras, A. and Sperotto, A. and Moreira Moura, G.C. and Drago, I. and Barbosa, R.R.R. and Sadre, R. and de Oliveira Schmidt, R. and Hofstede, R.J. 2010. *Attacks by "Anonymous" WikiLeaks Proponents not Anonymous.* Technical Report TR-CTIT-10-41, CTIT, University of Twente.
- Drago, I. and Pras, A. 2010. *Scalable Service Performance Monitoring.* In Proceedings of the 4th International Conference on Autonomous Infrastructure, Management and Security. AIMS'10. pp. 175–178.