UNIVERSITAT ROVIRA I VIRGILI

# EFFICIENT BITTORRENT-LIKE CONTENT DISTRIBUTION FOR CLOUD STORAGE SERVICES

**Rahma Chaabouni**

UNIVERSITAT
ROVIRA i VIRGILI

# Efficient BitTorrent-like Content Distribution for Cloud Storage Services

Rahma Chaabouni

DOCTORAL THESIS

2016

Rahma Chaabouni

# Efficient BitTorrent-like Content Distribution for Cloud Storage Services

Doctoral Thesis

Supervised by:
Dr. Pedro García-López
Dr. Marc Sánchez-Artigas

Department of Computer Engineering and Mathematics

UNIVERSITAT ROVIRA i VIRGILI

Tarragona

2016

UNIVERSITAT ROVIRA i VIRGILI
**Departament d'Enginyeria Informática i Matemátiques**
Av.Paisos Catalans 26,
43007 Tarragona, Spain
Tel: (+34) 977 558 745

I STATE that the present study, entitled "*Efficient BitTorrent-like Content Distribution for Cloud Storage Services*", presented by Rahma Chaabouni for the award of the degree of Doctor, has been carried out under our supervision at the Department of Computer Engineering and Mathematics of this university.

Tarragona, November 7, 2016

Doctoral Thesis Supervisors

[illegible] [illegible]

Dr. Pedro García-López Dr. Marc Sánchez-Artigas

*To the memory of my father,*
*the first to teach me.*

*To my beloved mother,*
*for her prayers for me.*

*To my dear husband,*
*for his support and patience.*

# Acknowledgment

Undertaking this PhD has been a life-changing experience for me that would not have been possible without the support and guidance I received from many people.

Foremost, I am deeply indebted to my advisers, Pedro Garcia-Lopez and Marc Sanchez-Artigas, not only for their great guidance and fruitful discussions but also for their confidence in giving me the opportunity to carry on this thesis. This work could not be possible without their continuous support and guidance. Thank you very much!

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.

Next, I would like to express my gratitude to all the former and current members of the AST research group at Rovira i Virgili University. I especially thank Cristian and Adrian for helping me settle down in Tarragona. I am also very grateful for the invaluable support received from two exceptional sisters that I met during my stay in Spain: Fatima and Reham, thank you for helping me feel like home. Special thanks to my Tunisian friends in Barcelona: Manel and Ines. I am lucky to have met you. Also, I would like to thank Imen, whose company in Tarragona has helped me a lot.

Last but certainly not least, I thank my family for their love, care, and support: my mother Najoua, my husband Hassen, my brother Ala and my stepparents: Sara and Moustapha. I sincerely hope that I have made you proud. This thesis is dedicated to you all.

# Contents

# Abstract

With the ever increasing Internet traffic, peer-to-peer (P2P) content distribution has emerged as an alternative to the traditional client-server model, especially with the recent bandwidth soar on the edges of the Internet. Data centers with limited bandwidth budget can benefit from the upload speed of the clients interested in the same content to improve the overall Quality of Service (QoS). This can be done by introducing a P2P protocol, BitTorrent for instance, when the load on a certain content becomes high.

The main challenge is to decide when is the best time to switch from the classic distribution protocol (HTTP) to BitTorrent, for each requested file. In fact, it is commonly assumed that BitTorrent is only efficient with big files and large sets of users, and that client-server protocols (including HTTP) perform better in the distribution of small files. However, to the best of our knowledge, there is no concrete analytic analysis of the transition point in efficiency between BitTorrent and HTTP. Our first contribution consists in the investigation of this transition point and the proposal of an algorithm that provides a simple switching strategy based on the QoS requirements of the system.

As a result from the introduction of BitTorrent, the data center will be faced with the challenge of managing its resources among clients that can be using different download protocols (HTTP and BitTorrent). To this extent, we calculate the amount of data center bandwidth needed to ensure a given ratio between the download times in HTTP and BitTorrent and propose a bandwidth allocation algorithm that decides the most suitable protocol for each case and provides the corresponding bandwidth allocations at the swarm level.

Nevertheless, the benefits that can be derived from the introduction of BitTorrent are tied with the number of simultaneous download requests of the same content. This number can be very limited in comparison with the number of separate download requests related to different files.To increase the contribution of the clients and benefit from the resources of peers involved in separate downloads, it is possible to make clients from different swarms cooperate with their spare bandwidth. We propose cross-swarm bundling to mitigate the problem of lack of simultaneous download requests. Cross-swarm bundling is the process of merging two swarms together into a single one. We prove that this technique can have a positive effect on the Quality of Experience (QoE), and present a methodology to implement bundling in data centers based on graph matching techniques.

All our proposals are evaluated using a trace of a Personal Cloud (Ubuntu One), and the results show that BitTorrent is can improve the performance of these systems when the cloud's available outgoing bandwidth is limited.

**Keywords** Content distribution, Personal Clouds, Online cloud storage, BitTorrent, QoS, QoE, Bandwidth allocation, Bundling.

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **IP** | Internet Protocol |
| **CDN** | Content Delivery Network |
| **QoS** | Quality of Service |
| **QoE** | Quality of Experience |
| **HTTP** | Hypertext Transfer Protocol |
| **SSL** | Secure Socket Layer |
| **TLS** | Transport Layer Security |
| **HTTPS** | HTTP over TLS, HTTP over SSL, or HTTP Secure |
| **FTP** | File Transfer Protocol |
| **P2P** | Peer-to-peer |
| **CPU** | Central Processing Unit |
| **MP3** | MPEG-1 and/or MPEG-2 Audio Layer III |
| **BT** | BitTorrent |
| **PC** | Personal Cloud |
| **URL** | Uniform Resource Locator |
| **SHA** | Secure Hash Algorithm |
| **SHA-1** | Secure Hash Algorithm 1 |
| **AES** | Advanced Encryption Standard |
| **U1** | Ubuntu One |
| **GMT** | Greenwich Mean Time |
| **Inc.** | Incorporation |
| **Ltd.** | Limited company |
| **S3** | Simple Storage Service |
| **HTML** | HyperText Markup Language |
| **RFC** | Request For Comments |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **VoD** | Video on Demand |

| | |
|---|---|
| **LAN** | Local Area Network |
| **CDF** | Cumulative Distribution Function |
| **Kbps** | Kilo-bits per second |
| **Mbps** | Mega-bits per second |
| **KBps** | Kilo-bytes per second |
| **KB** | Kilo-bytes |
| **MB** | Mega-bytes |
| **s** | Seconds |

# List of Symbols

| | |
|---|---|
| $\boldsymbol{s}$ | Swarm |
| $\boldsymbol{L_s}$ | Number of peers in the swarm $s$ |
| $\boldsymbol{f_s}$ | The file being downloaded by the peers in $s$ |
| $\boldsymbol{F_s}$ | Size of the file $f_s$ |
| $\boldsymbol{w_s}$ | Amount of cloud upload bandwidth allocated to $s$ |
| $\boldsymbol{d_{min,s}}$ | Download speed of the slowest peer in $s$ |
| $\boldsymbol{L_s}$ | Number of peers in $s$ |
| $\boldsymbol{D_s}$ | Aggregated download speed of all the peers in $s$ |
| $\boldsymbol{U_s}$ | Aggregated upload speed of all the peers in $s$ |
| $\boldsymbol{u_s}$ | Average upload speed of the peers in $s$ |
| $\boldsymbol{\eta_s}$ | Effectiveness of file sharing |
| $\boldsymbol{T_s^{http}}$ | Download time using HTTP for the swarm $s$ |
| $\boldsymbol{T_s^{pa}}$ | Download time for peer-assisted systems using the estimation of Kumar *et al.* [75] for the swarm $s$ |
| $\boldsymbol{T_s^{bt}}$ | Download time using BitTorrent for the swarm $s$ |
| $\boldsymbol{x_i(t)}$ | Rate at which seeds send bits to leecher $i$ at time $t$ |
| $\boldsymbol{r_i(t)}$ | Rate at which the leecher $i$ downloads 'fresh' content at $t$ |
| $\boldsymbol{\alpha_{bt}}$ | Overhead related to the start-up phase in BT transfers |
| $\boldsymbol{\tau}$ | Switching constraint |
| $\boldsymbol{W}$ | Maximum upload capacity of the data center |
| $\boldsymbol{Gain_s}$ | Gain ratio for $s$ |
| $\boldsymbol{Offload_s}$ | Offload ratio for $s$ |
| $\boldsymbol{switched_s}$ | State of the $s$ |
| $\boldsymbol{S_{HTTP}}$ | Set of swarms using HTTP as a download protocol |
| $\boldsymbol{S_{BT}}$ | Set of swarms using BitTorrent as a download protocol |
| $\boldsymbol{S}$ | Set of all the swarms, $S = S_{HTTP} \cup S_{BT}$ |
| $\boldsymbol{N_s}$ | Number of pieces in $f_s$ |

| | |
|---|---|
| $\boldsymbol{k_s}$ | Number of connections a peer in $s$ has |
| $\boldsymbol{t_{s_i}^{before}}$ | Expected end download time before bundling for $s_i$ |
| $\boldsymbol{t_{s_i,s_j}^{after}}$ | Expected end download time after $s_i$ and $s_j$ are bundled |
| $\boldsymbol{gain_{s_i}(s_j)}$ | Expected gain that $s_i$ would experience if it is bundled with $s_j$ |
| $\boldsymbol{gain_{s_i,s_j}}$ | Expected weighted gain when $s_i$ and $s_j$ are bundled |
| $\boldsymbol{Gain}$ | Objective matrix |
| $\boldsymbol{\Delta QoE}$ | Difference in download times before and after bundling |
| $\boldsymbol{\gamma}$ | Limit on the gain condition on the moment of bundling |

# 1 Introduction

In the last few years, we have witnessed a surge of Internet usage all over the world. With the ever increasing number of Internet users, the global Internet traffic in 2015 stood at 72.5 exabytes[1] per month, and it is predicted to surpass the zettabyte[2] threshold in 2016 and reach 2.3 zettabytes by 2020 [23]. This traffic relies mainly on content distribution, which represents the cornerstone of the Internet. It ranges from the distribution of web pages, to online video streaming through the delivery of large files. This ever increasing amount of traffic makes data delivery from the data centers to the end-users a challenging task.

With the traditional client-server model, the simultaneous requests of popular content can create bandwidth bottlenecks, which can affect the overall performance of the content provider and increase the download times for the end-users. Peer-to-peer (P2P) content distribution can present an attractive alternative especially with the recent bandwidth soar on the edges of the Internet. Data centers with limited bandwidth budget can benefit from the upload speed of the clients interested in the same content

[1] 1 exabytes = 1 million terabytes = $10^{18}$ bytes
[2] 1 zettabyte = 1 billion terabytes = $10^{21}$ bytes

to improve the overall Quality of Service (QoS). This can be done by introducing a P2P protocol, BitTorrent [24] for instance, when the load on a certain content becomes high. In fact, the efficiency of the BitTorrent protocol makes it especially suitable for files shared between a set of devices. In such scenarios, it is possible to benefit from the common interest of users in the same file and use their upload bandwidth to offload the data center from doing all the serving.

The main challenge is to decide when is the best time to switch from the classic distribution protocol (HTTP) to BitTorrent, for each requested file. Despite the large amount of research work dedicated to the study of BitTorrent, most of it is focused on the delivery of large volumes of data to a large set of end-users. As a matter of fact, it is commonly assumed that BitTorrent is only efficient with big files and large sets of users, and that client-server protocols (including HTTP) perform better in the distribution of small files [124, 125]. However, to the best of our knowledge, there is no concrete analytic analysis of this transition point in efficiency between BitTorrent and HTTP. We believe that it is essential to investigate this transition point in order to be able to select the best protocol for each file transfer. This can be translated in finding an answer to the following question:

**Question 1** *How to decide which protocol (HTTP or BitTorrent) is more suitable for each file transfer case?*

The evaluation of the efficiency of these protocols for each file transfer scenario can be made based on the current load on the data center and the characteristics of the files and the clients requesting it. This evaluation gives the data center the flexibility to adapt the distribution protocols based on its needs. As a result, the data center will have to manage its resources among clients that can be using different download protocols (HTTP and BitTorrent). With systems handling HTTP requests only, there are several policies to distribute the bandwidth between the different requests including: equal share, proportional to demand, etc. With BitTorrent only, there is also important related work in the field [96, 110]. However, there are no current policies for systems serving both types of clients. Thus, it is essential to define new policies to optimize the bandwidth usage while maintaining the QoS constraints, especially when the data center's resources are limited. This challenge can be formalized as the following question:

**Question 2** *How to balance the available bandwidth resources between the concurrent HTTP and BitTorrent swarms?*

In answering this question, the main goal is to minimize the share of data center's bandwidth reserved for each file transfer while taking into consideration the overall QoS constraints. To this extent, it is important to estimate the minimum amount of data center bandwidth needed to ensure a given QoS level in BitTorrent. Based on this value, the data center can decide to switch the download protocol for a given swarm from HTTP to BitTorrent if it (the data center) will save in bandwidth and if this change of protocol will not jeopardize the QoS constraint.

However, the benefits that can be derived from the introduction of BitTorrent are tied with the number of simultaneous download requests of the same content. This number can be very limited in comparison with the number of separate download requests focused on different files. Thus, to maximize the benefits from the BitTorrent swarms, it is important to engage these separate downloads in BitTorrent swarms. So the next challenge is the following:

**Question 3** *How to increase the contribution of the clients?*

To increase the contribution of the clients and benefit from the resources of separate downloads, is is possible to make clients from different swarms cooperate with their spare bandwidth. Swarms can be grouped in pairs where each swarm of the pair is committed to contribute with a part of its resources to boost the download process of the other swarm. This idea is inspired from the concept of *bundling* which is commonly used in BitTorrent systems [64].

Bundling is the process of grouping a set of content into a single file for download. Bundling was mainly used with BitTorrent swarms to mitigate the problem of availability in unpopular torrents [63, 65, 87, 129, 130]. In data centers, low content availability is not an issue: the data center is always available to serve the requested files. However, by introducing cross-swarm bundling, we aim at improving the Quality of Experience (QoE) of the users.

**Thesis contribution** In this thesis, we answer all the previously stated questions and provide the mechanisms and tools to integrate BitTorrent in data centers. We validate our proposal on online storage services (also referred to as Personal Clouds), through a trace of a real system. To summarize, the list of our contributions are:

- We conduct a comparative study between BitTorrent and HTTP. We first confute the general statement that BitTorrent is not effective for small files, based on a real experimental study. Then, we propose an analytic estimation of the distribution time in BitTorrent that takes into account the overheads related to the nature of the protocol. In addition, we introduce two general metrics to decide when it is better to use one protocol with respect to the other: the gain and the offload ratios. The gain measures the degree of improvement in terms of download time of BitTorrent relative to HTTP. The offload ratio quantifies the amount of data that can be offloaded if the peers adopt BitTorrent. We also propose a dynamic algorithm for the decision of the most appropriate download protocol. The algorithm uses simple parameters that can be collected by the system and predicts the efficacy of HTTP and BitTorrent for each case. The most suitable protocol is decided based on the predefined constraints. This algorithm is validated on a real trace of a Personal Cloud and it proves its efficiency in achieving important savings in terms of cloud bandwidth.

- We analyze the relationship between the amount of data center bandwidth allocated to a given group of clients and the resulting download time. Based on a fixed QoS constraint, we calculate the amount of seed bandwidth needed to ensure a given ratio between the download times in HTTP and BitTorrent. We also propose a dynamic algorithm which uses simple parameters that can be collected by the system and evaluates the efficacy of using HTTP and BitTorrent as a distribution protocol for each requested file.

Based on the load on the data center and the predefined switching constraints, the algorithms decides the most suitable protocol for each case and provides the corresponding bandwidth allocations at the swarm level. This algorithm can be applied in data center-based content distribution systems to achieve important improvements in the overall QoS.

- We propose cross-swarm bundling as a solution to engage more users in BitTorrent transfers. As far as we know, we are the first to provide evidence that bundling can be useful to diminish the download time, even in a very adverse scenario where most of files are small. More specific to our problem is the investigation of the benefits of inflating swarms with HTTP users or even of the merge of two HTTP users to create a BitTorrent swarm. We also study the efficiency of cross-swarm bundling in data centers through an average case analysis and identify the cases where it is positive. Finally, we validate our proposal using a real trace of a Personal Cloud system and compare the different approaches that can be deployed to deliver content from the cloud to end users. Our results demonstrate that when the load is high, bundling can improve significantly the QoE of users without increasing the data center bandwidth.

**Thesis organization** The rest of the thesis is organized as follows:

- **Chapter 2: Background** details the concepts used in this thesis, mainly the BitTorrent protocol and Personal Clouds. We also present in this chapter the trace used in the validation of the proposed algorithms.

- **Chapter 3: State of the Art** gives an overview of the research work carried in relation with the subjects of this thesis and illustrates the key differences between the contributions of the thesis and the previous works.

- **Chapter 4: Protocol Decision Strategy** is dedicated to answering the first question raised in this chapter. It presents a complete comparison between two content distribution protocols: HTTP and

BitTorrent and a study on the efficiency of each protocol in distributing small files to small sets of clients. The chapter presents also the switching algorithm and a complete application scenario of protocol switching in Personal Clouds.

- **Chapter 5: Bandwidth Allocation Strategy** is related to the second question. It presents an efficient algorithm to distribute the data center's bandwidth between the different concurrent swarms, especially when there is a mixture of both BitTorrent and HTTP clients.

- **Chapter 6: Cross-Swarm Bundling** responds to the third question. It presents the concept of cross-swarm bundling, evaluates the efficiency of this approach on the QoE, and presents a methodology to select the pairs of swarms to bundle.

- **Chapter 7: Conclusions** concludes the thesis. It includes a summary of the findings, a list of the papers published during the thesis research and the planned future work.

# 2 Background

In this chapter, we provide the necessary concepts and definitions to properly understand the rest of the thesis. We start by introducing the most common paradigms for content distribution. Next, we present a basic content distribution scenario and provide an estimation of the download time using the client-server and the peer-assisted paradigms. After that, we present a specific peer-to-peer protocol: BitTorrent which is the main protocol used in the rest of the thesis. Next, we describe Personal Clouds (online Cloud storage services) which represent the application scenario for the algorithms proposed in this thesis. Finally, we introduce *Ubuntu One*, a Personal Cloud system operated by *Canonical Ltd*, and explore a trace of this system to study the clients and file characteristics.

## 2.1 Content Distribution Paradigms

In this section, we present an overview of the most common content distribution paradigms currently used to distribute content across the Internet today. We mainly cover in this section the client-server, the peer-to-peer and the peer-assisted models.

### 2.1.1 The Client-Server Model

The client-server model distinguishes between the providers of a resource or service, called servers, and the service requesters, called clients. Clients are responsible for initiating communication sessions by sending requests to servers. In the client-server paradigm, clients can communicate with servers only, and they do not interact with each other. Servers are responsible for responding to their clients by acting on each request and returning results. One server generally supports numerous clients.

The client-server model is a core network computing concept of the Internet. However, it presents a major performance drawback when the server is overloaded with too many clients. In this case, the server's performance will experience improper functioning, increased latency or even total shutdown [27].

#### The Hypertext Transfer Protocol (HTTP)

Several web technologies and protocols are built around the client-server model. The *Hypertext Transfer Protocol (HTTP)* [44] is commonly used for downloading/uploading data all over the Internet. HTTP is an application-layer protocol that represents the foundation of data communication for the World Wide Web. It is mainly used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web.

HTTP is a request-response, stateless protocol that follows the client-server paradigm. Each HTTP message is either a request or a response. RFC[1] 7231 [46] defines the semantics of HTTP messages and state the different request methods. These methods indicate the purpose for which the clients have made the request and what is expected by the client as a successful result. Table 2.1 presents some of the standardized methods that are commonly used in HTTP, along with a short description of their usage.

The HTTP client starts by initiating an HTTP request. The server processes the request and sends a response back. Any type of data can be

[1]A Request For Comments (RFC) is a type of publication from the *Internet Engineering Task Force* (IETF) and the *Internet Society* (ISOC), the principal technical development and standards-setting bodies for the Internet.

[2]Uniform Resource Identifier (URI) is a string of characters used to identify a resource

**Table 2.1:** Some of the commonly used HTTP request methods.

| Method | Description |
|---|---|
| **GET** | Requests data from a specified URI |
| **HEAD** | Same as GET but returns only HTTP headers |
| **POST** | Submits data to be processed to a specified resource |
| **PUT** | Requests that the enclosed entity be stored under the specified URI[2] |
| **DELETE** | Deletes the specified resource |
| **OPTIONS** | Returns the HTTP methods that the server supports |

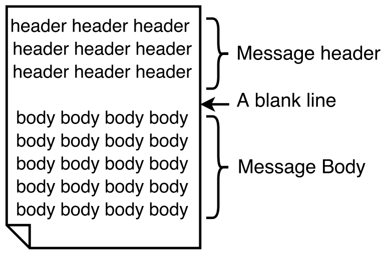


**Figure 2.1:** HTTP message format

sent via HTTP. The type of content is specified in the header section of request and response messages. As a matter of fact, an HTTP message consists of a message header and an optional message body, separated by a blank line, as illustrated in Figure 2.1. RFC 7230 [45] states that most HTTP communication consists of a retrieval request (GET) for a representation of some resource identified by a URI. It defines the message exchange as follows:

1. A client sends an HTTP request to a server in the form of a request message. The message begins with a request line that includes the method (GET in this case), URI, and protocol version, followed by other header fields. This header is followed by an empty line that separates it from the message body. Finally, the message contains the payload body, if any.

2. A server responds to a client's request by sending one or more HTTP response messages, each beginning with a status line that includes the protocol version, a success or error code, and textual reason

```
Client request:

     GET /hello.txt HTTP/1.1
     User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
     Host: www.example.com
     Accept-Language: en, mi

Server response:

     HTTP/1.1 200 OK
     Date: Mon, 27 Jul 2009 12:28:53 GMT
     Server: Apache
     Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
     ETag: "34aa387-d-1568eb00"
     Accept-Ranges: bytes
     Content-Length: 51
     Vary: Accept-Encoding
     Content-Type: text/plain

     Hello World! My payload includes a trailing CRLF.
```

**Figure 2.2:** A typical message exchange example between a client and a server for a GET request on the URI: “http://www.example.com/hello.txt” (source RFC 7230 [45]).

phrase. This line is followed by some header fields containing server information, resource meta-data, and representation meta-data. The response message also contains an empty line that indicates the end of the header section, and a message body containing the payload body.

**HTTPS** HTTPS [103] (also called *HTTP over TLS*, *HTTP over SSL*, and *HTTP Secure*) is a variant of HTTP that adds a layer of security on the data in transit through a Secure Socket Layer (SSL) or Transport Layer Security (TLS) protocol connection.

### Content Delivery Networks

A *Content Delivery Network* (CDN) is a distributed network of proxy servers deployed in multiple datacenters located in different geographical areas. CDNs were mainly designed to speed the delivery of content of websites with high traffic and global reach. To this extent, a CDN stores a cached version of its content in multiple geographical locations, called

*Points of Presence* (PoPs). Each PoP contains a number of caching servers responsible for content delivery to visitors within its proximity. The closer the PoP is to the user geographically, the faster the content will be delivered.

The three key components of a CDN architecture are: the content provider, the CDN provider and the end-users [95]. The content provider is the original server than has the original content. The CDN provider is a proprietary organization or company that provides the infrastructure facilities to content providers. The end-users are the clients who request content from the content provider's website.

*Akamai* [5, 33] is the most prominent CDN player of the world as of today. It operates with more than 216,000 servers in over 120 countries and within more than 1,500 networks around the world[3]. Akamai delivers daily Web traffic reaching more than 30 terabits per second, corresponding to nearly 3 trillion Internet interactions every day.

### 2.1.2 The Peer-to-Peer Model

A *peer-to-peer* (P2P) system has a completely decentralized resource usage and self-organization [113]. In these systems, each entity in the network has equivalent functionality: it can play both the roles of client and server. This helps in reducing bottlenecks and enables each entity to contribute with a part of its own resources (such as storage and bandwidth capacities, CPU power, etc.). P2P architectures have been employed for a variety of different application categories [8], including mainly: distributed computation [7, 114] and content distribution [3, 83].

P2P file sharing (called also P2P content distribution) applications are probably the most popular P2P applications. Without the need for central coordination, a peer can download data from other peers and, at the same time, collaborate with its own bandwidth to deliver data to them. This technology has evolved and gained in popularity thanks to the increasing Internet bandwidth.

P2P file sharing became popular in 1999 with the introduction of *Napster* [104]. This system was mainly used to share digital audio files, typically songs, encoded in MP3 format. Napster was the first system

[3]Akamai, facts & figures https://www.akamai.com/us/en/about/facts-figures.jsp

to recognize that requests for popular content do not need to be sent to a central server but instead could be handled by many peers that have the requested content [86]. This system was based on a centralized server that indexed the users and their shared content [121]. When someone searched for a file, the server's role was limited to providing a list of all the peers that have copies of that file. In 2001, Napster was shut down due to copyright infringement.

After Napster was shut down, *Gnutella* [3, 106] emerged as an alternative that allowed users to download different files other than music, such as movies and games. Unlike Napster, Gnutella is a decentralized network that distributes both the search and download capabilities among peers. These Gnutella peers form an overlay network by forging point-to-point connections with a set of neighbors [107]. The network uses flooding as the mechanism to send queries across the network [86]. When a peer receives the flood query, it sends a list of all content matching the query to the originating peer.

Nowadays, the *BitTorrent* protocol is considered as a leading P2P content distribution protocol. This protocol has been used widely for delivering large files like movies, books and TV shows. A detailed description of this protocol is available in the following sections.

### 2.1.3 The Peer-Assisted Model

In this thesis, we refer to peer-assisted content distribution systems as partially centralized systems that rely mainly on P2P content distribution while having a central server to guide and coordinate the peers and help in the distribution process when needed. Thus, the peer-assisted model represents the combination of both the client-server and the P2P models. This model resembles the client-server in that there is sill a central server where data are stored and which guarantees that all the files are always available for download. However, in the peer-assisted model, the peers that are downloading the files also assist in redistributing them. The main difference between this model and the pure P2P one relies in the existence of a central server. This server is especially important when the peers cannot satisfy the demand: there are only a few peers in the system, or the available peers do not have a full copy of the content, etc.

An example of peer-assisted systems is *CloudAngels* [115]. It is a peer-

**Table 2.2:** Comparison between the content distribution paradigms.

| | **Client-server** | **CDN** | **Pure P2P** | **Peer-assisted** |
|---|---|---|---|---|
| **Data Flow** | Centralized | Centralized | Distributed | Hybrid |
| **Control** | Centralized | Centralized | Distributed | Centralized |
| **QoS** | Guaranteed | Guaranteed | Best effort | Guaranteed |
| **Scalability** | Limited | High | High | High |
| **Cost** | High | High | Low | Medium |
| **Distance from end-users** | Large | Small | Small | Small |

assisted bulk-synchronous content distribution service proposed by Sweha *et al.* that aims to improve average distribution times. The system uses dedicated servers, called *angels*, that offer their storage and bandwidth capacities to cache and forward data to the clients.

The peer-assisted model can be also applied in video streaming for Video on Demand (VoD). In peer-assisted VoD distribution, videos are streamed from dedicated VoD servers as well as from other online users. Cha *et al.* [17] explored the efficacy of P2P techniques on YouTube[4] through a trace-driven analysis. Their results showed that the users can reduce the server workload by 41% when they share videos while they are watching them. A similar study [67] applied to the *MSN Video* [91] service proved that peer-assisted VoD can trim the server bandwidth by 58.2%.

The peer-assisted model can be applied to CDNs [47], too. *Akamai NetSession* [131] is an example of a peer-assisted CDN that relies on both the servers' infrastructure of Akamai and resources from the clients' machines. This system uses HTTP/HTTPS to download content from edge servers and a swarming protocol for downloads from peers. NetSession was capable of offloading a large fraction (more than 70%) of the traffic to peers while offering good performance and high reliability [131].

## Summary

To summarize, each of the previously described paradigms can be efficient for a different application scenario. Table 2.2 gives a comparative overview between these paradigms. The client-server paradigm is characterized with centralized control and data flows. These systems exhibit limited scalability with the increasing number of requests and the cost of

[4]YouTube https://www.youtube.com

this scalability is generally high.

CDNs are an evolution of the client-server model, created with the objective of improving the download time. They rely on the replication of popular data in multiple servers located across the globe. Each download request is fulfilled by the nearest server, which makes the distance from the content provider to end-users much smaller. The P2P model has a completely decentralized self-organization and resource usage. These systems are characterized by their high scalability, since each arriving peer contributes with part of its resources while downloading the data. However, the decentralized nature of these systems can affect the Quality of Service (QoS). Finally, the peer-assisted paradigms is a combination of both client-server and P2P. It is based on hybrid (both centralized and distributed) data flow and a centralized control. The QoS is guaranteed by the central server while scalability is ensured thanks to the clients' contributions.

## 2.2 The Content Distribution Scenario

One of the goals of this thesis is to provide means to determine the most appropriate content distribution approach/protocol to make the downloads faster for each file transfer case. In this context, we consider the following content distribution model. We suppose that there are two main entities in the system: the data center (or content provider) and the clients interested in downloading content. The content provider is responsible for delivering the requested content to the corresponding clients.

Figure 2.3 represents the file transfer scenario: for each file $f_s$ being transferred, we assume that there are $L_s$ clients that wish to download it within a small window of time. We denote by $s$ the set of these clients. Each client $c_i$ in $s$ has an upload speed $u_i$ and a download speed $d_i$. We denote by $d_{min,s} = \min\limits_{c_i \in s} d_i$ the download speed of the slowest peer in $s$, and by $u_s = \sum\limits_{c_i \in s} \frac{u_i}{L_s}$ the average upload speed of the peers in $s$. We denote by $w_s$ the amount of data center's upload bandwidth allocated to $s$ and by $F_s$ the size of the file $f_s$.

Currently, there is a large number of applications that can allow such transfers. We focus here on two main models: the traditional client-server

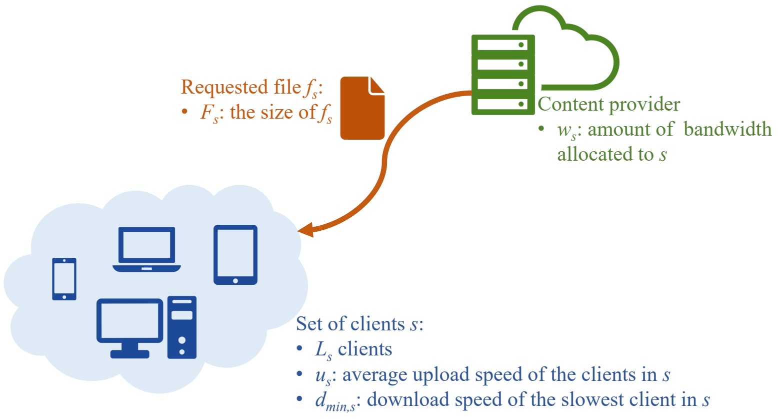

Requested file $f_s$:
• $F_s$: the size of $f_s$
Content provider
• $w_s$: amount of bandwidth allocated to $s$
Set of clients $s$:
• $L_s$ clients
• $u_s$: average upload speed of the clients in $s$
• $d_{min,s}$: download speed of the slowest client in $s$

**Figure 2.3:** The file transfer scenario.

model, and the peer-assisted model.

### 2.2.1 Download Time in the Client-Server Model

In the client-server model, it is possible to estimate the data transfer time of a given file, provided that the transfer bandwidth is known, as follows:

$$transfer\ time = \frac{size\ of\ file}{bandwidth}. \tag{2.1}$$

The estimated transfer time is equal to the ratio between the size of the file being transferred and the bandwidth of the slowest link between the source host and the destination host.

In the case $f_s$ is distributed via HTTP, the distribution time $T_s^{http}$ is limited by the download speed of the slowest peer $d_{min,s}$ or the provider's bandwidth $w_s$ divided equally between the $L_s$ clients. It can be formulated as follows:

$$T_s^{http} = \frac{F_s}{\min\left\{d_{min,s}, \dfrac{w_s}{L_s}\right\}}. \tag{2.2}$$

Nowadays, with the bandwidth evolution, the download capacities of the clients are getting higher. If we suppose that the download speed of the clients is not a bottleneck $\left(d_{min,s} > \frac{w_s}{L_s}\right)$, the download time becomes $T_s^{http} = \frac{F_s}{w_s} L_s$, which grows linearly with the number of clients $L_s$.

### 2.2.2 Download Time in the Peer-Assisted Model

Kumar *et al.* [75] presented a fluid model for peer-assisted content distribution based on fluid replication. Using this model, they proposed an estimation of the download time in peer-assisted systems, as follows:

$$T_s^{pa} = \frac{F_s}{\min\left\{d_{min,s}, \dfrac{w_s + u_s\, L_s}{L_s}, w_s\right\}}, \tag{2.3}$$

where $T_s^{pa}$ is the minimum time needed to distribute a file $f_s$ of size $F_s$ to $L_s$ peers. This time depends on the download speed of the slowest peer $d_{min,s}$, the aggregated upload bandwidth of all the nodes divided equally between all the $L_s$ peers, and the upload bandwidth of the seed(s). The authors presented in their paper a complete proof of the download time. The proof was organized into the following four exhaustive cases depending on the parameter that may be responsible for the transfer bottleneck:

1. Case A: $d_{min,s} \leq \min\left\{\frac{w_s + L_s\, u_s}{L_s}, w_s\right\}$ and $d_{min,s} \leq \frac{u_s\, L_s}{L_s - 1}$: In this case, the download speed of the peers is limited by the download bandwidth of the slowest peer $d_{min,s}$.

2. Case B: $d_{min,s} \leq \min\left\{\frac{w_s + L_s\, u_s}{L_s}, w_s\right\}$ and $\frac{u_s\, L_s}{L_s - 1} \leq d_{min,s}$: In Case B, the transfer is limited by the maximum speed at which a peer can get data from the other peers, that is $\frac{u_s\, L_s}{L_s - 1}$.

3. Case C: $\frac{w_s + L_s\, u_s}{L_s} \leq \min\{d_{min,s}, w_s\}$: The transfer bottleneck in this case is limited by the aggregated upload speed of the network $w_s + L_s\, u_s$ divided equally between the $L_s$ peers.

4. Case D: $w_s \leq \min\left\{d_{min,s}, \frac{w_s + L_s\, u_s}{L_s}\right\}$: In this case, the upload bandwidth of the seed $w_s$ is the maximum limit at which each peer can download "fresh" content.

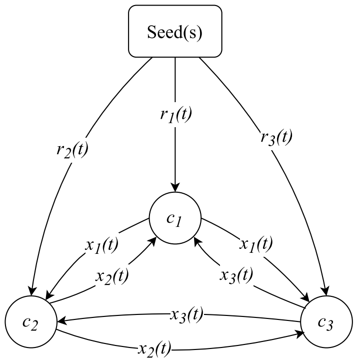


**Figure 2.4:** General distribution scheme structure: client $c_i$ ($i \in \{1, 2, 3\}$) downloads "fresh" data at the rate $r_i(t)$ from the seed(s). The data is replicated later to the other 2 clients at a rate $x_i(t)$, where $x_i(t) \leq r_i(t)$.

For each of the cases listed above, the authors constructed a seeding rate profile $r_i(t)$ which denotes the bit rate at which the seeds send pieces to a client $c_i$ at time $t$.
The adopted distribution scheme is the following: as soon as a client $c_i$ begins to receive data from the seed, it replicates it to each of the other $(L_s - 1)$ peers at a rate $x_i(t)$, where $x_i(t) \leq r_i(t)$, as shown in Figure 2.4. For each case, the distribution scheme consists of $L_s$ application-level multicast trees, each rooted at a specific seed, passing through one of the peers and terminating at each of the $L_s - 1$ other clients.

The authors also presented in the same paper and analytic estimation of the seeding rate $r_i(t)$. This rate depends on the time $t$, the file size $F_s$, the upload speed of the seeds $w_s$, and upload and download speeds of the clients in $s$.

$$r_i(t) = \begin{cases} \frac{u_i d_{min,s}}{\eta_s u_s L_s}, & \underline{\textit{Case A}} \\ \frac{u_i - \eta_s u_s L_s}{L_s - 1} + d_{min,s}, & \underline{\textit{Case B}} \\ \frac{u_i - \eta_s u_s L_s}{L_s - 1} + \frac{w_s + \eta_s u_s L_s}{L_s}, & \underline{\textit{Case C}} \\ \frac{u_i w_s}{\eta_s u_s L_s}. & \underline{\textit{Case D}} \end{cases} \tag{2.4}$$

## 2.3 The BitTorrent Protocol in a Nutshell

The BitTorrent protocol [24] is a P2P protocol that relies on the bandwidth of peers to quickly disseminate files in a distributed fashion. The basic idea in BitTorrent is to divide the file into smaller pieces of equal size. Each peer interested in downloading the file has to connect to several other peers simultaneously and download the different pieces from them.

To facilitate the communication between peers, BitTorrent uses a centralized server called tracker. Any peer that wants to download the file has to contact the tracker first. The tracker returns a random list of peers that have that file (or parts of it). Using this list, the downloader can establish a connection with other peers and request the pieces it does not have from them.

Pieces represent the transfer unit in BitTorrent. When a peer receives a new piece, it becomes a source of that piece for other peers. Pieces are typically downloaded non-sequentially and are rearranged into the correct order by the BitTorrent client. The BitTorrent client is also responsible for monitoring the pieces list and checking which pieces are needed and which ones can be uploaded to other peers.

The first BitTorrent client was released in July 2001 by the protocol's inventor *Bram Cohen*. Since then, many clients have been made available for a wide variety of computing platforms and operating systems. Among the most famous clients, we can list: *Transmission* [119], *qBittorrent* [117], *Vuze* [10], *Deluge* [29], *BitComet* [11], and *$\mu$torrent*[12], the official client released by *BitTorrent, Inc.*

### Terminology

The main terms used throughout this thesis[5] are:

- **BitTorrent Client**: A BitTorrent client is a computer program that implements the BitTorrent protocol and which allows users to transfer data via BitTorrent.

[5]A complete glossary of the BitTorrent protocol can be found the official BitTorrent client web page http://help.bittorrent.com/customer/en/portal/articles/179175-glossary.

- **Peer**: A peer is an instance of a BitTorrent client running on a computer on the Internet to which other clients can connect and transfer data.
- **Seed**: A seed is a peer that has a full copy of the data.
- **Leecher**: A leecher is a peer that does not have a full copy of the data.
- **Swarm**: A swarm is the collection of all the peers (seeds and leechers) downloading the same data.
- **Tracker**: A BitTorrent tracker is a server that keeps track of the seeds and leechers in the swarm. Periodically, all the peers report their state to the trackers and receive information about other peers in the swarm. The tracker is not directly involved in the actual distribution of the file, it only assists in the communication between peers.
- **Piece**: The files transferred using BitTorrent are split into small pieces (also called chunks) of equal size (typically 512 KB).
- **Block**: Each piece is split into smaller blocks (typically 16 KB). Blocks represent the transmission unit on the network.
- **Peer set**: The peer set is the list of peers sharing the file known by the client.
- **Active peer set**: The active peer set is the list of peers to which the BitTorrent client is currently sending data [98].
- **Torrent**: A torrent can refer to either a `.torrent` meta-data file or all files described by it, depending on the context. The `.torrent` meta-data file contains the details about the files to be downloaded (names, sizes, checksums of all the pieces), along with the address(es) of the corresponding tracker(s).
- **Free riders**: Free riders are peers that consume resources and download data without contributing to the swarm.
- **Interested/Not interested**: Peer A is interested in peer B when peer B has pieces that peer A does not have. Conversely, peer A is not interested in peer B when peer B only has a subset of the pieces of peer A [78].

- **Choked/Unchoked**: Peer A chokes peer B when peer A decides not to send data to peer B. Conversely, peer A unchokes peer B when peer A decides to send data to peer B [78].
- **Rarest pieces**: The rarest pieces are the pieces that have the least number of copies in the peer set.
- **Availability**: The availability is a metric that measures the number of complete copies of a torrent in the corresponding swarm. The availability is an indication of the rapidity of the download. The higher the availability is, the faster the download is likely to be. When the availability is low, pieces are scarce and peers have to spend more time downloading the file.

### Algorithms

**Rarest First Algorithm** The rarest-first algorithm is mainly used for the selection of pieces to download. Knowing the number of copies of each piece in its peer set, each peer defines a set of the rarest pieces. This set is updated each time a copy of a piece is added to or removed from its peer set [78]. Each peer selects the next piece to download randomly from its rarest pieces set.

Bram Cohen outlines two exceptions to the rarest first peer selection strategy [24]. The first exception is when the download starts. At that time, when the peer has nothing to upload, it is important to get a complete piece as quickly as possible. Since rare pieces' download would be slower [24], the first pieces to download by each peer are selected at random. This is called the *random first policy*. The second exception is the *end-game mode*. This mode starts when all blocks have been received or requested by a peer, that is at the end of the download. To make download faster, in end-game mode, the peer requests all blocks not yet received to all the peers in its peer set that have the corresponding blocks.

**Choking Algorithm** The choking algorithm is also known as the tit-for-tat algorithm and it is mainly used for peer selection. It aims at guaranteeing a reasonable level of upload and download reciprocation to penalize free riders. We remind that choking is a notification that no data will be sent until unchoking happens, but the local peer can still download from the choked peers. We suppose that each peer $c_i$ (seed or leecher) unchokes a fixed number $n$ of remote peers (by default $n = 4$).

The choking algorithm is executed periodically (every 10 seconds) by each peer as follows: First, the local peer $c_i$ orders interested remote leechers according to the rates at which it received data from them, and ignores leechers that have not sent any data in the last thirty seconds. The $n - 1$ leechers with the highest rates are unchoked via a *regular unchoke*. In addition, every 30 seconds, an interested peer is chosen at random to be unchoked via an *optimistic unchoke*.

### Operations

In a BitTorrent transfer, there are two main entities: the original publisher and the downloaders. The original publisher is typically the file owner and is the one that decides to make that file available for download via BitTorrent. The downloaders have to run a BitTorrent client and get the meta-data `.torrent` file in order to download the file. Each of these entities has to follow a few steps, as described bellow, in order to ensure the distribution of the file.

**Creating and publishing torrents** When a publisher decides to share some content, he has to create the meta-data `.torrent` file. This file is a bencoded[6] dictionary that can be created using most BitTorrent clients and which contains mainly two keys: *announce* and *info*. The first key is related to the URL of the tracker. The second key maps to a dictionary that contains information about the file being shared, including: the name and size of the file, the length of pieces (in bytes), and the list of SHA-1 hashes of all the pieces, etc.

After creating the meta-data file, the initial publisher has to make it available to the interested downloaders. This can be done directly via emails, if the publisher wants to share his file with a limited set of peers, or using a BitTorrent portal. These sites allow users to host their `.torrent` files and make them available for other users to find them.

[6]Bencode is the encoding used by BitTorrent for storing and transmitting the meta-data information related to a torrent file.

**Downloading torrents and sharing files** The entities interesting in downloading a file using BitTorrent have to get first a copy of the meta-data `.torrent` file. As mentioned above, this file can be obtained directly from the publisher or downloaded from the Internet through special websites. Once the `.torrent` is downloaded, it can be opened using a BitTorrent client and the peer join the swarm.

To join the swarm, the BitTorrent client contacts the tracker to get a list of the IP addresses of other peers in the swarm in order to build the initial peer set. This list typically consists of 50 peers chosen at random in the list of peers currently involved in the torrent [78]. Using this list, the client establishes direct communication with these peers and starts downloading (and eventually uploading) the pieces that constitute the file. The exchange of pieces among peers is governed by the previously described rarest first and choke algorithms.

## 2.4 Personal Cloud Systems

Personal Clouds have emerged lately as user-centric solutions that provide easy management of the users' data. According to *Statista*[112], in 2014, there was over 1.1 billion people worldwide using Personal Clouds[7]. This represents about 15% of the total world population[8] and 37% of the population using the Internet[9]. This popularity makes these systems worth investigating in order to improve further their performance and ensure their scalability with the increasing number of users.

### 2.4.1 Overview and Examples

A Personal Cloud is a term generally used to refer to a file hosting service that allows its users to store, synchronize and share content over

[7]Number of Personal Cloud users from 2014 to 2019 http://www.statista.com/statistics/499558/worldwide-personal-cloud-storage-users

[8]The world population in 2014 was about 7.2 billion according to the Population Reference Bureau http://www.prb.org/pdf14/2014-world-population-data-sheet_eng.pdf

[9]The number of people using the Internet in 2014 was about 3 billion people according to the International Telecommunication Union http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf

the Internet. Garcia *et al.* [53] propose the following definition: *"The Personal Cloud is a unified digital locker for our personal data offering three key services: Storage, Synchronization and Sharing. On the one hand, it must provide redundant and trustworthy cloud data storage for our information flows irrespective of their type. On the other hand, it must provide syncing and file exploring capabilities in different heterogeneous platforms and devices. And finally, it must offer fine-grained information sharing to third-parties (users and applications)."*

Personal Clouds have gained their popularity via consumer products. Currently, there a wide range of products offering different advantages to the consumers. The following list includes some of the most popular of these products.

- **Google Drive**: Google Drive [57] is the freeware Personal Cloud launched in 2012 by *Google*[10]. Google Drive is available in different platforms and it offers also an integrated free web-based office suite (Google Docs [56], Google Sheets [58] and Google Slides [59]).
- **Dropbox**: Dropbox [38] is a very popular Personal Cloud launched in 2007 which offers storage, synchronization and sharing capabilities to its clients. Dropbox operates on different operating systems and can be accessed via web browsers and desktop and mobile apps. In March 2016, Dropbox announced that its number of users hit half a billion users[11].
- **OneDrive**: OneDrive [92], formally called *SkyDrive*, is the Personal Cloud launched by Microsoft[12] in 2007. OneDrive can be accessed via a web browser or desktop and mobile devices.
- **Box**: Box [13] is a Personal Cloud that was founded in 2005. It offers a freemium business model to provide cloud storage and file hosting for personal and professional accounts.
- **Ubuntu One**: Ubuntu One was a file hosting service launched in 2009 by Canonical[13] that allowed users to store data in the cloud. In

[10]Google Inc. https://www.google.com
[11]ZDnet: Dropbox hits half a billion users http://www.zdnet.com/article/dropbox-hits-half-a-billion-users/
[12]Microsoft Corporation https://www.microsoft.com/
[13]Canonical Ltd. http://www.canonical.com/

**Table 2.3:** Comparison of some Personal Clouds features.

| | Google Drive | Dropbox | Box | OneDrive | Ubuntu One |
|---|---|---|---|---|---|
| **Storage nature** | Private | Public | Private | Private | Public |
| **Free storage limit** | 15 GB | 2 GB | 10 GB | 5 GB | 5 GB |
| **Max file size** | 5 TB | 20 GB | 250 MB | 10 GB | 5 TB |
| **Real-time collaboration** | Yes | No | No | Yes | No |
| **Server-side encryption** | ? | Yes | Yes | ? | No |
| **Deduplication** | No | Yes | ? | ? | Yes |
| **P2P syncing** | No | Yes (LAN) | No | No | No |

April 2014, Canonical announced that Ubuntu One would be shut down at the end of July 2014.

To attract a large number of users, Personal Cloud providers try to extend their systems with different functionalities. Table 2.3 presents a comparison of the characteristics and functionalities offered by the above listed products. Based on the nature of the storage back-end, Personal Clouds can be public when they use third party data centers (such as Amazon), or private when they use their own infrastructure and do not outsource the storage task [52]. Among the listed products, only Dropbox and Ubuntu One use external storage servers. They both use *Amazon Simple Storage Service (S3)*.

The amount of free storage initially offered to the regular clients differs from one provider to the other. Currently, Google Drive offers the bigger quota of free storage (15 GB) which is an important feature that can attract many users. In addition to the storage limit, some providers put a constraint on the maximum file size that can be stored. The relatively low limit on file size put by Box (250 MB) can affect the popularity of this system, since the users will not be able to store large files (long videos for instance) in their personal accounts.

Real-time collaboration is an important feature which allows multiple users to edit a file at the same time. Currently, only Google Drive and OneDrive offer this functionality to their users through their web-base office suites. Security is another important factor in Personal Clouds. The use of these services implies a certain level of trust between the clients and the service providers. To gain this trust, some Personal Clouds (like Dropbox and Box) claim to encrypt the users' data stored in their servers. Nevertheless, server-side encryption is not enough when the server controls also the encryption keys. That is why it is recommended that users encrypt their data locally before transmitting them to the server.

To improve their performance and reduce costs, Personal Clouds can deploy a deduplication mechanism. When a user is uploading a file that already exists in the storage nodes, the Personal Cloud only creates a logical relation between the file and the user instead of re-uploading the existing file. This technique can be applied to different scopes (that is across all files in the system, across user's files, etc.), depending on the privacy policy [52].

P2P syncing is another feature that can be deployed to reduce the bandwidth cost of Personal Clouds. P2P file syncing is the ability to keep two or more files identical in different locations without resorting to a central service [52]. Currently, P2P syncing is only deployed in Dropbox and it is only limited to synchronization between machines on the same Local Area Network (LAN).

### 2.4.2 Architecture

Despite the rapid growth of Personal Clouds, many of the commercial products are proprietary and their architecture and algorithms are invisible to the research community[53]. However, there are a few elements that can be qualified as essential to the functioning of these systems. These elements are presented in Figure 2.5 that presents a general architecture of a Personal Cloud. This architecture is inspired from Dropbox's white paper for business security [1]. For the sake of simplicity, we do not present here the authentication and encryption modules.

A typical Personal Cloud follows a 3-tier architecture consisting of: (i) tier i: the user interfaces, (ii) tier ii: the processing and the notification services and (iii) tier iii: the meta-data and storage services, as follows:

- **User interfaces:** The services offered by Personal Clouds can be utilized and accessed by physical clients through a number of interfaces. Typically these interfaces include web interfaces (accessed through web browsers), desktop applications and mobile apps.
- **Notification service:** The notification service is dedicated to monitoring whether or not any changes have been made to the users' accounts. Whenever a change to any file takes place, the client is

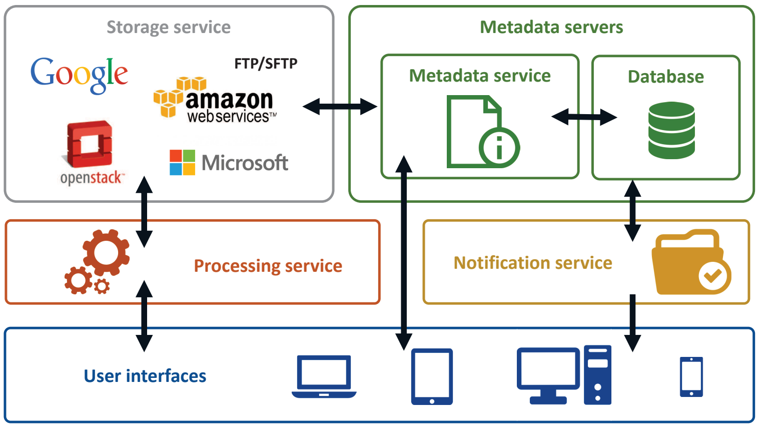


**Figure 2.5:** General architecture of Personal Clouds.

notified in order to synchronize these changes. This service is only used to notify the clients and it is neither responsible for storing nor transferring files or meta-data.

- **Processing service:** The processing service is responsible for processing the files and ensuring their delivery to the end-users. To download a file, the client sends an HTTP GET request to the processing service. The latter verifies the existence of the file in the storage nodes and the file is transferred using the HTTP protocol.
- **Meta-data service:** The meta-data servers contain all the meta-data information related to clients and files. Clients' meta-data include basic account and user information, such as: email address, name, and device names. Files' meta-data group information about users' data such as file names and types. The meta-data service is generally equipped with a local database where all the meta-data is stored.
- **Storage service:** The storage service or storage back-end refers to the physical locations where the users' data are stored. It can be local, in the form of local storage servers accessed via FTP, or external, provided by a third party like Amazon or Google.
  To meet the users' requirements, the storage service must guarantee

the availability of data files stored by users. This can be achieved by adding redundancy to multiple servers.

Security is an important feature in Personal Clouds. Currently, there is no standard proposal for security in these systems. Each commercial product has its own security scheme incorporated in its architecture. For example, Dropbox uses SSL/TLS to create a secure tunnel for data transfer and Advanced Encryption Standard (AES) to encrypt the data at rest.

## 2.5 The Ubuntu One Trace

This thesis aims to provide efficient contributions that can be applied in a real-world Personal Clouds. To this extent, all the proposed algorithms are validated using a trace of a real Personal Cloud system: Ubuntu One. In this section, we present this system, describe the collected trace and study the users' and files' characteristics.

### 2.5.1 The Ubuntu One System

Ubuntu One (U1) was a file hosting service operated by Canonical that allowed users to store data in the cloud. U1 was first launched in May 2009 in beta version[14]. In April 2014, Canonical announced that the cloud storage and synchronization features would be shut down at the end of July 31 of 2014. Like most Personal Clouds, U1 followed a 3-tier architecture that consists of clients, a synchronization service and of storage and meta-data services. Canonical only owned the infrastructure for the meta-data service. The actual contents of file transfers were stored separately in Amazon S3 [60].

### 2.5.2 Trace Description

The trace was provided by Canonical in the context of the *CloudSpaces* project[15]. The logs were collected during 30 days from their meta-data servers located in London, based on the behavior of their users.

[14]The Ubuntu One Blog: http://voices.canonical.com/ubuntuone/2009/05/
[15]FP7 CloudSpaces Project: http://www.cloudspaces.eu

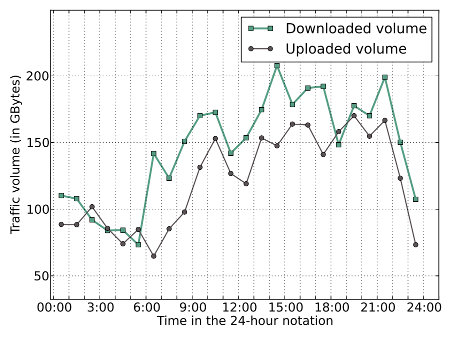
```
time  operation file_id file_size user_id bandwidth
0.0 GET 1543575822 8189 704866851 0.00198063377558
0.009 GET 15455023 815 1223956618 8.80730388516e-05
0.013 PUT 3457758605 62 4178692664 4.25992850293e-05
0.017 GET 3247597761 46541 1526014448 0.0052131731689
0.043 GET 3246172185 23376 1910373866 0.00514021001153
0.056 PUT 1425838301 638195 2662396472 0.0202350615187
0.146 GET 3146421807 377567 3899301018 0.0562618672846
0.154 PUT 176541977 332 4178692664 0.0002926246516
0.155 PUT 346240330 1622 4178692664 0.00103746461522
0.156 PUT 1541002195 4099406 3420193392 0.00500194245457
0.16 PUT 390089416 398 2311519970 1.33771191207e-05
0.161 PUT 3982622940 117 2311519970 3.12513710003e-06
0.176 PUT 3816660408 1305 1960464888 0.00256607212979
0.189 PUT 52868563 1243 4178692664 0.00063459163559
0.205 PUT 1615006207 4096 4049028426 0.000356344645136
0.207 GET 2807404967 39730 3448366816 0.00750881501985
0.214 PUT 2214900784 8890 4046205577 0.00262076187725
0.233 GET 2832935497 62 2821005149 1.37187488673e-05
0.258 PUT 975132975 1060956 4045930323 0.00647498136511
0.263 GET 3543677586 84985 4174733981 0.0159386453845
```

**Figure 2.6:** Snapshot of the first 20 PUT and GET operations in the trace.

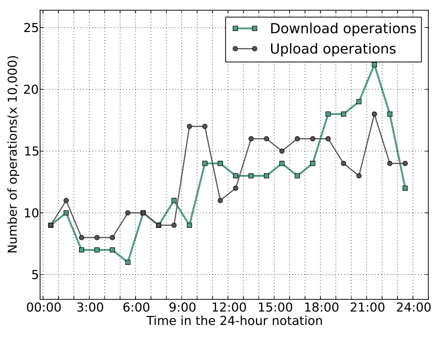
We filtered the original trace and focused on the upload and download operations that were performed during a random day of the trace (January $21^{st}$, 2014). For that day, $6,331,131$ operations performed by $33,257$ distinct client on $4,095,057$ unique files were logged.

Each line of the filtered trace represents an upload or download operation performed by one user on a given file. For the sake of privacy, files and user real identifiers are presented in the form of unique hash codes. For each operation, several information were collected, including: the timestamp (in seconds), the type of operation ("GET" or "PUT"), the hash and size (in bytes) of the file in question, the user's hash identifier and the corresponding upload/download bandwidth (in KBps). Figure 2.6 presents a snapshot of the first 20 file upload and download operations in the trace.

Figure 2.7 shows the total uploaded and downloaded volume along with the total number of upload and download operations measured per hour during the whole day. The hours are logged according to the Greenwich Mean Time (GMT). We notice that the peak with the highest number of upload and download operations corresponds to the hour between 21:00 and 22:00 GMT.

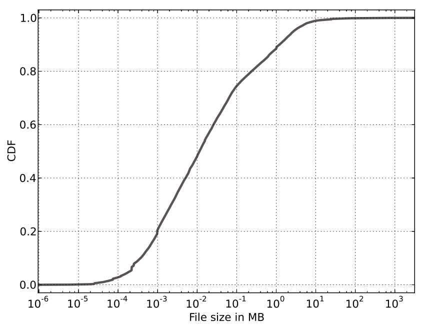


(a) Volume

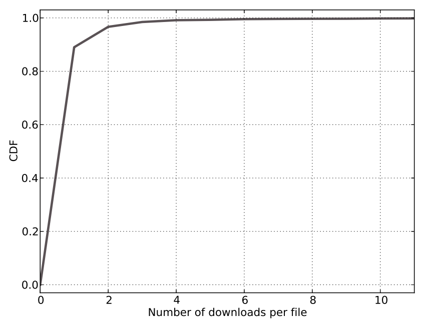


(b) Operations

**Figure 2.7:** Total uploaded and downloaded volume and the number of upload and download operations per hour during the day January 21$^{st}$, 2014 for the U1 system.



(a) File sizes



(b) Downloads per file

**Figure 2.8:** CDF of the file sizes and the number of downloads per file in the sample.

**Clients and Files Characteristics** We focus on this peak hour with the highest number of upload and download operations. Since we aim to manage efficiently the upload speed of the cloud, we filter the one-hour sample to keep only the **225, 514 download operations** related to **173, 756 distinct files**. The number of unique users involved in this sample is **81, 083 users**.

Figure 2.8a presents the cumulative distribution function (CDF) of the file sizes in the sample. We notice that most of the files are small. The average file size in the sample is about 1 MB, 988.26 KB to be precise.

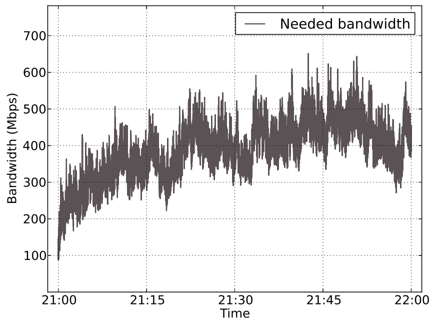


**Figure 2.9:** Needed upload bandwidth over time during the peak hour.

Figure 2.8b plots the CDF of the downloads per file. We notice that about 68.62% of the operations correspond to single downloads. Single downloads are operations related to files downloaded only once. These files account for about 89% of the total files downloaded between 21:00 and 22:00. This means that only 31.38% of the operations correspond to multiple downloads of the same file and only 11% of the files were downloaded more than once.

**Bandwidth Requirements** Focusing more on the peak hour, we calculate how much upload bandwidth should be provided by the cloud in order to satisfy the demand of all the requesting peers. We consider the default protocol used to distribute the files in Personal Clouds, which is HTTP.

To estimate the bandwidth needs, we went through the trace tracking the active swarms at each timestamp and summing up all the download capacities of the active peers. With each new download request, we updated the amount of data left to be downloaded by each peer. Once a peer has finished downloading the file, it was removed from the active peers list. The download times were calculated according to (2.2).

The resulting amount of needed cloud upload bandwidth is presented in Figure 2.9. This figure will be useful later to set a potential limit on the cloud seed's bandwidth when evaluating the efficiency of the proposed algorithms. It shows that the total need in cloud bandwidth does not exceed 650 Mbps. So, when evaluating the algorithms, it would be better to vary the cloud limit $W \in ]0, 650[$ in order to measure the effect of the seed's capacity on the algorithm's performance.

# 3 State of the Art

This chapter covers the related work addressed around the different problems that motivate this thesis. It is organized into sections according to the different treated topics. Each section is concluded with a small comparative study of the related work and the contribution of the thesis.

## 3.1 Content Distribution Paradigms

In this section, we review some of the related work on the content distribution paradigms related to this thesis. This study includes peer-assisted content distribution and the BitTorrent protocol.

### 3.1.1 Peer-Assisted Content Distribution

In this thesis, we refer to peer-assisted content distribution systems as partially centralized systems that rely mainly on P2P content distribution while having a central server to guide and coordinate the peers and help in the distribution process when needed.

These systems have been studied thoroughly in the literature. Karaginannis *et al.* [73] explore the impact of peer-assisted content distribution systems on the content providers, the Internet Service Providers (ISPs) and the end users. They demonstrate that these systems can provide significant benefits for the content provider and end users, but have an adverse impact on the ISPs. These results are further confirmed in [105], where the authors prove that the peer-assisted approach can yield substantial performance gains and capacity savings compared to a pure client/server system and improve the reliability compared to a pure P2P scenario.

Peer-assisted systems can be used in a wide range of applications. A prominent example is Amazon's standard offering for BitTorrent content distribution in the Amazon Simple Storage Service (S3) [6]. Apart from the standard REST and SOAP APIs, it is possible to retrieve objects stored in S3 using BitTorrent. Palankar *et al.* [94] evaluate the use of Amazon S3 services for Science Grids. They pay special attention to the use of BitTorrent in S3 as a cooperative cache that can reduce costs when transferring large amounts of data. In a similar context, Sweha *et al.* [115] proposed to use dedicated servers (called angels) to accelerate peer-assisted content distribution. These angels are not the original content providers, nor are they interested in the content. Their main purpose was to increase the total upload capacities of the swarms by using their storage and bandwidth capacities to cache and forward parts of the requested files to the interested peers.

Peer-assisted CDNs have been proposed to reduce the content distribution costs. Michiardi *et al.* [90] presented a general framework and a prototype implementation of peer-assisted CDNs. Akamai NetSession [131] is a commercial system that was proven to offload $70 - 80\%$ of the traffic to the peers without a corresponding loss of performance or reliability.

Peer-assisted content distribution is also applied for online data backup [118] and Video On Demand (VoD) [67, 116, 122, 128]. In peer-assisted VoD distribution, videos are streamed from dedicated VoD servers as well as from other online users. Cha *et al.* [17] explored the efficacy of P2P techniques on YouTube[1] through a trace-driven analysis. Their results showed that the users could reduce the server workload by 41% when they shared videos while they were watching them. A similar study [67] applied to the *MSN Video* [91] service proved that peer-assisted VoD could trim the server bandwidth by 58.2%.

[1]YouTube https://www.youtube.com

### 3.1.2 The BitTorrent Protocol

BitTorrent [24] is one of the most used protocols for peer-assisted content distribution. This protocol had attracted the researchers' attention and several studies were dedicated to the measurements, analysis [68, 89, 99, 101, 123] and modeling [61, 75, 80, 100] of the BitTorrent ecosystem. Most of the measurements of BitTorrent involved real torrents and have proven that BitTorrent is an effective inexpensive tool for content distribution [68]. Some studies [25, 30, 31, 32, 34] were more focused on concrete aspects of the protocol. Others aimed mainly to provide a measurement data that can be useful to model BitTorrent systems [99].

In the context of measuring and analyzing BitTorrent, the authors in [68] monitored a large torrent involving thousands of peers during five months. Through the collected data, the authors evaluated the performance of the algorithms used in BitTorrent. Their results confirm that BitTorrent represents an effective inexpensive means for content distribution. Another prominent measurement study was carried out by Pouwelse *et al.* [99]. The authors focused on four different issues: availability, integrity, flash-crowd handling, and download performance. The authors aimed with their study to provide a measurement data that can be useful to model BitTorrent systems.

Several models were proposed for BitTorrent. One the most popular models is the one of Qiu *et al.* [100]. The authors presented a fluid model and studied the scalability, performance and efficiency of BitTorrent-like systems. In this thesis, we take advantage of their model to estimate the effectiveness of file sharing. This metric was thoroughly studied in their paper and an analytic model was proposed to estimate it. The work presented by Kumar *et al.* [75], which presents an important pillar of this thesis, is related to the modeling of peer-assisted systems in general. Based on fluid-model arguments, the authors derived explicit expressions for the minimum distribution time of a general heterogeneous peer-assisted file distribution system. These expressions are generic and depend on the file size, the seeds' upload rates and the leechers' upload and download rates. The proposed formulas were validated through comparison with experimentally measured distribution times using BitTorrent.

Improving BitTorrent's performance is also an interesting research topic. A lot of research work has been dedicated to improve the incentive mechanism [80, 98], improve the peer selection strategy [76, 127] and fight against the problem of free-riders [22, 72].

All this collaborative work has resulted in an ever increasing popularity of the protocol with different application domains. BitTorrent has proven its efficiency in the distribution of virtual machine images [109, 102] with a 30x speedup over traditional approaches. This protocol was also implemented in a university scenario [40] and helped reducing the number of its servers from 20 to only 2 central servers, sufficient to keep their 6,500 workstations updates. BitTorrent has been also proven to be efficient in delivering Video on Demand (VoD) [9, 26, 67]. It has also been used to distribute large data inside data centers [69, 70]. Finally, Twitter [49], Facebook [41] and even the United Kingdom's government [120] use BitTorrent.

BitTorrent is generally considered efficient for big files and big swarms only. To the best of our knowledge, there were only a few studies that compared the performance of BitTorrent with small files. Wei *et al.* [124, 125] compared experimentally the BitTorrent and FTP protocols. They used the comparison to evaluate the use of a decentralized architecture for distributing data in grid systems. They came to the conclusion that BitTorrent outperforms FTP when the file size is greater than 20 MB and when the number of nodes is greater than 10. However, these limits are only relative to their specific deployment setup and cannot be generalized in other settings such as regular clients in a wide area network. In [43], this comparison was extended to include the HTTP protocol and a wider range of file sizes. However, this work is also very specific to the experiments settings and the conclusions cannot be generalized. Kumar *et al.* provided in [75] the tools for a generic comparison between the client-server and peer-assisted paradigms. They proposed an estimation of the download times for each paradigm. Even though their estimation did not consider the overheads related to the BitTorrent, it can extended and used as a base for a generic comparison (which is one of the scopes of Chapter 4).

## Progress beyond state of the art

Currently, most of the studies related to BitTorrent focused on the efficiency of this protocol in distributing large files to big swarms. In these circumstances, BitTorrent has proven its efficiency in reducing the load on the original content provider and improving the download times for end users. On the other hand, it is assumed that client-server protocols, HTTP for example, are more efficient than BitTorrent when the files and/or the number of downloaders are small. Nevertheless, the switching point between BitTorrent and HTTP remains unclear. Wei *et al.* [124, 125] proposed 20 MB as a file size limit and 10 as a swarm size limit. However, these limits are only relative to their concrete deployment and cannot be generalized.

In Chapter 4, we start by proving through experimentation that even with swarms with less than 10 clients and files smaller than 20 MB, BitTorrent can achieve lower download times than HTTP. This confutes the general assumption that BitTorrent is only efficient with big files/swarms and confirms the need for a generic study of both protocols in order to better define the switching point between them. In the same chapter, we extend the formulas proposed by Kumar *et al.* [75] to consider the overheads related to the nature of the protocol. We also propose two metrics to quantify the performance of BitTorrent compared to HTTP in terms of download time and bandwidth usage. Finally, we study the criteria that can be considered in the definition of the switching point and present a simple algorithm that manages the download protocols based on the predefined QoS constraints.

## 3.2 Bandwidth Allocation Problem

We have presented in the previous section various research works focused on how to efficiently distribute content to a set of users. In this section, we focus on works that aim to improve the bandwidth allocation of a server, given a restricted bandwidth budget.

In general, independent from the application fields, Sharma *et al.* [110] classified the bandwidth allocation strategies into two main categories. The first category groups static, simple strategies such as uniform allocation, best-effort or proportional to demand. The second category is related to dynamic strategies that constantly adjust the allocation in response to fine-grained client-perceived performance, so as to optimize the performance or cost objectives of the content distributor. Bandwidth allocation problems can be mapped onto classic optimization problems with varying complexity. For instance, [4] sought optimal bandwidth allocation in wireless multimedia networks. The problem was modeled as an NP-complete multiple-choice knapsack problem [111]. Lagrangean relaxation procedure was used to develop an efficient heuristic algorithm for this problem. The Portfolio problem [88] is another common problem onto which resource allocation problems can be mapped [74, 93]. Zhu *et al.* [132] addressed the problem of bandwidth allocation for different service classes in heterogeneous wireless networks. This problem was modeled as an optimal control problem [15] and solved using Nash equilibrium. Finally, linear programming's interior point primal dual method [126] was proposed in [71] as a solution for network allocation in wireless cellular and ad hoc networks.

Bandwidth allocation strategies depend on the protocol being used. In this thesis, we focus mainly on two protocols: HTTP and BitTorrent, and in this section we review some of research works related to bandwidth allocation techniques with these two protocols. HTTP servers can deploy several techniques to distribute their resources among the different simultaneous clients, such as traffic shaping [66, 81, 85] and scheduling algorithms [14]. *Antfarm* [96] and *V-Formation* [97] are two examples of systems using dynamic bandwidth allocation strategies in BitTorrent-like swarms. Antfarm is a content distribution system that measures a swarm's response curve to seeder bandwidth in order to optimize its uploads among competing swarms. This system needs to actively measure swarm dynamics and uses an off-band protocol to motivate users to report performance data that is later used to do the actual allocation. On the other hand, V-Formation evaluates the benefit of available bandwidth to competing consumers and targets toward a global allocation of bandwidth that maximizes the aggregate download bandwidth of consumers.

**Progress beyond state of the art** In this thesis, we focus on the bandwidth allocation problem in systems dedicated to deliver files to different clients that can be using either the HTTP or BitTorrent protocols. This means that at the same time, the data center is required to serve both kind of clients: the ones that use HTTP and the ones that use BitTorrent. To the best of our knowledge, there is no related work that deals with both kind of protocols. But, there have been some research works that addressed the bandwidth allocation in HTTP servers and peer-assisted systems based on BitTorrent separately.

In Chapter 5, we study the relationship between the amount of bandwidth allocated to a swarm of clients and the resulting download time. Based on a fixed quality of service constraint, we calculate the amount of seed bandwidth needed to ensure a given ratio between the download times in HTTP and BitTorrent. Moreover, we propose a dynamic algorithm that decides the most suitable protocol for each case and provides the corresponding bandwidth allocations at the swarm level.

## 3.3 Bundling in BitTorrent

Several extensions have been proposed in the literature in order to increase the performance of BitTorrent. One of these extensions is bundling. Bundling consists in grouping a set of contents into a single file for download. Peers download a bundle that contains both the desired files along with some other files that make up the bundle.

Bundling in BitTorrent has been reported to increase the availability of unpopular files. Menasche *et al.* [87] performed a measurement study on real BitTorrent swarms and proved the efficiency of bundling in improving the availability of unpopular torrents and reducing the download times when publishers were unavailable. Another measurement study, conducted by Han *et al.* [64, 65], focused on the prevalence of file bundling in the BitTorrent ecosystem. The authors noticed that over 70% of the monitored torrents contained multiple files which proved that bundling was very common.

Bundling can be *static* [16] when a pre-determined set of files are grouped together by the publisher, or *dynamic*[63] if peers are assigned complementary content to download at the time they decide to download a particular file. Though static bundling is easy to implement, it may result in wasted downloads as every peer has to download the entire bundle. Dynamic bundling offers more flexibility since it can adapt to the current state of the publishers which might help in avoiding wasted downloads [130].

To improve further the benefits of bundling, different download strategies were proposed and evaluated. Lev-tov *et al.* [79] proposed a dynamic file selection and download strategy. This strategy allowed peers to dynamically select whether to collaborate or not when they were interested in downloading only a small subset of the bundled files. Zhang *et al.* [129, 130] introduced the idea of "super bundle" which consisted of a large collection of files. Clients who requested individual files from this super bundle were instructed to download additional ones. The set of additional files was adjusted by the tracker to dynamically adapt the bundles to the relative file availability.

A prominent work in the field of dynamic bundling is the one of Han *et al.* [63] where the authors proposed nine different bundling strategies to dynamically bundle swarms in pairs. Two different categories of strategies were proposed: attribute-based and access-history based. Attribute-based strategies considered the attributes of torrents, such as the title or tag, and based on the similarity of these attributes, several torrents could be bundled into a single one. Access-history based strategies, however, were based on the access pattern of a swarm and considered dynamic parameters such as popularity, availability, and life cycle.

To select the pairs of swarms to group from the large set of available combinations, Han *et al.* use the *maximum weight matching algorithm* [39, 50]. In graph theory, a *matching* is a subset of edges such that none of the selected edges share a common vertex. With respect to a weighted graph, a *maximum weight matching* is a matching for which the sum of the weights of the matched edges is as large as possible. The first polynomial time algorithm for maximum matching was proposed in 1965 by Edmonds [39] and subsequently improved by Gabow and others [48, 50]. Currently, there are several libraries that can find the maximum weight matchings for dense graphs in time $O(n^3)$ ($n$ is the number of nodes in the original graph).

**Progress beyond state of the art** In current literature, bundling have been limited to grouping two or more BitTorrent swarms into a single one. To the best of our knowledge, no one have previously studied the effects of grouping other types of swarms. For instance, if we suppose that a set of clients downloading simultaneously the same file using HTTP can be called an *HTTP swarm*, then we can confirm that nobody has studied the effects of bundling HTTP swarms together or even an HTTP swarm with a BitTorrent one.

In Chapter 6, we evaluate bundling on swarms that use the same download protocol (all the swarms are using HTTP or all the swarms are using BitTorrent) and also swarms that use different protocols (some swarms are using HTTP and some others are using BitTorrent). Moreover, we go beyond the classic scope of bundling (which is to mitigate the problem of availability in unpopular torrents [87, 63]), and reveal other benefits of bundling, focusing mainly of the effects of bundling on the download time.

## 3.4 Personal Clouds

Personal Clouds refer generally to Dropbox[2]-like file hosting services that allow their users to store, synchronize and share content over the Internet. Personal Clouds have become very popular in the last years. They have attracted also the researchers' attention and there has been several studies related to the data characteristics and usage patterns in these systems.

The first studies on Personal Cloud systems were carried out by Drago *et al.* [37, 36]. In [37], they presented a characterization of Dropbox including typical usage, traffic patterns, and possible performance bottlenecks, and compared in [36] the performance of 5 popular providers. Drago *et al.* observed the daily activity patterns of the users and have detected that the service usage is time-based with significant peaks in the upload and download traffic. The files involved in this traffic followed a long-tailed distribution with a large percentage of files under 1 MB in size [35]. In the same context, Gracia *et al.* [60] studied a one-month long trace of Ubuntu One[3], a Personal Cloud system provided by Canonical. They reveal the internal architecture of the system, the core components of the system, as well as the interactions with Amazon S3 (the outsourced data storage service). The authors confirmed the daily activity patterns of the users and stated that 90% of the files involved in the upload/download traffic are smaller than 1 MB. Another measurement example [84] studied the characteristics of the data stored in a Personal Cloud system for campus students and revealed that 99% of the files are smaller than 16 MB, and that most of upload/download requests are for small files. Garcia *et al.* [53, 54] went beyond the classic measurement studies, and proposed

[2]Dropbox: https://www.dropbox.com/
[3]The Ubuntu One Blog: http://voices.canonical.com/ubuntuone/2009/05/

*StackSync*, an open-source Personal Cloud framework that provides scalable file synchronization and sharing among users. Their goal was to provide an open-source system that can serve as a reference architecture for Personal Clouds.

Personal Clouds offer to their clients data storage that can be automatically synced across multiple devices, and also shared among a group of users. To minimize the network overhead, current commercial cloud storage services employ delta encoding, data compression, and other mechanisms when transferring updates across users [36, 37]. Li *et al.* [82] proposed another optimization to mitigate the excessive traffic usage when transmitting frequent and short data updates. The authors deployed a middleware between the user's file storage system and a cloud storage client application that instantiated a *"SavingBox"* folder that replaced the default sync folder used by the cloud storage application. Frequent and short data updates were detected and batched in the SavingBox, and then pushed to the cloud storage application after a certain delay. In a similar context, Gonçalves *et al.* [55] investigated a trace of Dropbox and found that a large fraction of Dropbox downloads was associated with same content shared by multiple devices. Their proposal to introduce network caches could recover 92% of the costs of serving avoidable downloads.

In an attempt to reduce the load on the central servers, Dropbox deploys *LAN sync* [28], a feature that speeds syncing between machines on the same Local Area Network (LAN). With LAN syncing, Dropbox looks for the new file on the LAN first, bypassing the need to download the file from Dropbox servers, thus speeding up the syncing process considerably.

Finally, it is important to mention here that *BitTorrent Sync* [42, 108] (Now *Resilio Sync*[4]) cannot be considered as a Personal Cloud, as the users' data are not stored in the Cloud. Instead, it is a P2P file synchronization tool that relies on the BitTorrent protocol to synchronize data between the users' devices.

**Progress beyond state of the art** In this thesis, we investigate the effects of using BitTorrent in Personal Clouds. We start by measuring the effects of using BitTorrent to distribute files that are shared between a set

[4]In early 2016, Resilio Inc. was spun out of BitTorrent Inc. https://getsync.com/about/

of devices. In such scenarios, it is possible to benefit from the common interest of users in the same file and use their own upload bandwidth to offload the cloud from doing all the serving. In Chapter 5, we prove that BitTorrent can be effective in decreasing outbound traffic and improving the Quality of Service (QoS) of the system, despite the small size of data flows and swarms. In Chapter 6, we focus on a new dimension, the Quality of Experience (QoE), and set as our main goal the reduction of the delay experienced by devices to get the necessary files and deltas to be “in sync” again. To achieve this goal, we propose to “inflate” the swarms by grouping a set of diverse contents into a single `.torrent` file.

# 4 Protocol Decision Strategy

In this chapter, we propose a strategy to decide which protocol (HTTP or BitTorrent) is more suitable for each file transfer scenario. We start by proving through an experimental example that BitTorrent can outperform HTTP in the delivery of small files. Next, we propose an accurate estimation of the download time with BitTorrent. This estimation takes into consideration the delays that can be caused by the start-up phase and the lack of available pieces. After that, we propose two metrics to study the trade-offs between the HTTP and BitTorrent protocols. These metrics are the gain and the offload ratios. The gain ratio measures the improvements in term of download time, and the offload measures the amount of data that can be offloaded from the central server using BitTorrent. Next, we propose a simple algorithm that decides based on the predefined QoS constraints which protocol is more suitable for each file transfer. Finally, we validate the proposed algorithm on a real trace of a Personal Cloud and we prove that it can achieve important savings in terms of cloud bandwidth.

*The results presented in this chapter are published in [18] and [20]*

## 4.1 Introduction

This chapter aims at providing simple means that can help data centers decide which protocol should be used for each served file. The chapter is mainly dedicated to answering the first question raised in the introductory chapter, which is: *"How to decide which protocol (HTTP or BitTorrent) is more suitable for each file transfer case?"*.

To answer this question, we need to study the trade-offs between both protocols. In general, it is assumed that BitTorrent is only efficient with big files and big swarms, and that client-server protocols performs better in the distribution of small files [124, 125]. However, to the best of our knowledge, no concrete analytic comparison was made on this matter, and all the conclusions were drawn from experimental results that cannot be generalized. Thus, a generic study of both protocols is essential to make a wise decision about which protocol to use.

In this context, it is necessary to provide accurate estimations of the download times for each approach. An accurate estimation of BitTorrent should take into account the overheads that can be associated with the use of this protocol for small files, such as: the start-up phase overhead, and the overhead related to the lack of available pieces. These overheads are evaluated in this chapter and an accurate estimation of the download time with BitTorrent for small files is provided. This estimation is used later to quantify the gain and offload ratios. The gain ratio measures the gain in terms of download time that can be obtained using BitTorrent instead of HTTP, while the offload ratio measures the amount of data that can be offloaded from the content provider with the use of BitTorrent. The gain and offload ratios are two key parameters in deciding which download protocol is more suitable for each file transfer. They can be used to evaluate the efficiency of each protocol based on the predefined Quality of Service (QoS) constraints.

The remainder of the chapter is organized as follows: Section 4.2 investigates the trade-offs between HTTP and BitTorrent. It contains an accurate estimation of the download time using BitTorrent, along with two metrics to quantify the performance of this protocol compared to HTTP in terms of download time and bandwidth usage. In Section 4.4, we present the protocol decision strategy: we study the constraints that can be considered in evaluating the protocols, and propose an algorithm for managing the download protocols. This algorithm is validated in Section 4.2 in a real-world scenario using a trace of a Personal Cloud. The results prove that use of BitTorrent can reduce significantly the costs while maintaining the performance constraints. Finally, Section 4.5 concludes the chapter.

## 4.2 Trade-offs Between HTTP and BitTorrent

In this section, we study the trade-offs between the HTTP and BitTorrent protocols. We start by comparing through experimentation the distribution times for small files and swarms. We prove that, despite the general assumptions, BitTorrent can outperform HTTP even when the files are small and the swarms are composed of a few clients. Next, we propose an estimation for the download time using BitTorrent that is valid for small files/swarms, and which takes into consideration the overheads related to the nature of the protocol. Finally, we present two metrics that measure the trade-offs between HTTP and BitTorrent: the gain and offload ratios. The first measure the improvement in download times that can result from using BitTorrent instead of HTTP. The second measures the amount of data offloaded from the central server through the use of BitTorrent.

### 4.2.1 BitTorrent Can Be Efficient for Small Files

In general, it is commonly assumed that BitTorrent is only efficient with big files and big swarms and that client-server protocols perform better in the distribution of small files [124, 125]. However, to the best of our knowledge, no concrete analytic comparison was made on this matter, and all the conclusions were drawn from experimental results that cannot be generalized.

To confute this assumption, we run a simple experiment that consists in distributing small files (1, 5 and 10 MB) from a central server to small sets of clients (2, 3, 4 and 5 clients). Each file is sent using HTTP and BitTorrent separately. We use the following common ADSL bandwidth settings: each client $c_i$ has an upload bandwidth $u_i = 1$ Mbps, a download bandwidth $d_i = 2$ Mbps and the bandwidth allocated by the content provider to each transmitted file is $w_s = 5$ Mbps. The measured downloads time (in seconds) and the amount of data contributed by the clients (in MB) are reported in Table 4.1.

We notice that even though the files are small, there are some cases when BitTorrent downloads are faster that HTTP. We also notice that when the number of peers in the swarm increases, the improvements in download time using BitTorrent are more important. For instance, with 4 clients downloading a 1 MB file, BitTorrent is $\approx 6\%$ faster than HTTP. An extra client in the swarm ($L_s = 5$) can even improve the download to be $\approx 22\%$ faster than HTTP. Another important advantage of the use of BitTorrent is the amount of data contributed by the peers. This amount is quite important even when the files are small. Starting from just 2 peers, we notice that the contribution ranges from 11% (for $F_s = 1$ MB) to exceed 30% (for $F_s = 10$ MB). The peers' contributions is even more important when the number of peers is higher. These experimental results confute the general assumptions and confirm the need for a generic study of both protocols to make a wise decision about which protocol to use for each file transfer scenario.

### 4.2.2 Download Time with BitTorrent

We present an estimation of the download time with BitTorrent, optimized for small files/swarms. In this estimation, we reuse $T_s^{pa}$, which was first proposed by Kumar *et al.* in [75], and extend it to consider the overheads related to the BitTorrent protocol.

#### Limitations of $T_s^{pa}$

One of the limitations of (2.3), the estimation of the distribution time in peer-assisted systems, proposed by Kumar *et al.* [75], is that it does not take into consideration the overheads that peer-assisted systems may present compared to the client-server ones. For large files, the download times are generally long and the small protocol overheads can be neglected. However, they cannot be ignored for the small ones, for which the download time is in the order of a few seconds.

To illustrate the important role that this overhead plays in the distribution of small files, we run several experiments distributing a small file

**Table 4.1:** Measured download times for small files using HTTP and BitTorrent. The seed's bandwidth is limited to 5 Mbps. The clients are homogeneous, each having an upload and a download speed of 1 and 2 Mbps, respectively. The measured download times are in seconds and the data from peers are in MB.

| $L_s$ | $F_s = 1$ MB | | | | $F_s = 5$ MB | | | | $F_s = 10$ MB | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $T_s^{http}$ | $T_s^{bt}$ | $T_s^{http} - T_s^{bt}$ | **Data from peers** | $T_s^{http}$ | $T_s^{bt}$ | $T_s^{http} - T_s^{bt}$ | **Data from peers** | $T_s^{http}$ | $T_s^{bt}$ | $T_s^{http} - T_s^{bt}$ | **Data from peers** |
| **2** | 4 | 5.51 | -1.51 | 0.23 | 20 | 21.52 | -1.52 | 2.9 | 40 | 42.06 | -2.06 | 6.08 |
| **3** | 4.8 | 5.47 | -0.67 | 0.80 | 24 | 21.69 | 2.31 | 6.02 | 48 | 42.73 | 5.27 | 11.97 |
| **4** | 6.4 | 6.03 | 0.37 | 1.57 | 32 | 23.06 | 8.94 | 7.84 | 64 | 42.83 | 21.17 | 17.6 |
| **5** | 8 | 6.25 | 1.75 | 1.64 | 40 | 24.05 | 15.95 | 11.59 | 80 | 44.68 | 35.32 | 23.64 |

**Table 4.2:** Estimated versus experimental distribution times with BitTorrent for a 1 MB file.

| Number of clients | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Estimated time (s) | 4 | 4 | 4 | 4 |
| Experimental time (s) | 5.51 | 5.47 | 6.03 | 6.25 |
| Absolute error (s) | 1.51 | 1.47 | 2.03 | 2.25 |
| Relative error (%) | 37.75 | 36.75 | 50.75 | 56.25 |

(1 MB) to several clients. We consider swarms whose size range from 2 to 5 clients and use the same bandwidth settings as in the experiments used in Table 4.1. For each experiment, we measure the experimental download times and compare them to the estimated ones using (2.3). Also, we calculate the corresponding absolute and relative errors.

Table 4.2 groups the results of the experiments. It shows that even though the absolute error between the measured and estimated download times is small (a few seconds), it represents an important error percentage compared to the experimental time and can exceed 50% in some cases. This confirms the need for a more accurate estimation that includes the protocol overheads.

### Adding the BitTorrent Overheads

The $T_s^{pa}$ formula lacks mainly two types of overheads, each related to a different phase of BitTorrent: (i) the overhead related to the start-up phase and (ii) the overhead related the problem of piece availability in the download phase.

**Overhead related to the start-up phase** Before starting the download with BitTorrent, there are a few steps that each leecher needs to perform: First, the leecher has to get and read the `.torrent` file that contains all the meta-info data about the requested content. And then, it needs to contact the tracker(s) to get a list of other peers sharing or downloading the same file. Finally, after locating and connecting to the peers, the BitTorrent client can start exchanging pieces with them.

All these steps are specific to the BitTorrent protocol and they create an extra overhead, that we call the start-up phase overhead. This overhead depends on the set of peers returned by the tracker, and the moment at which the remote peers decide to unchoke the local peer [77].

We experimentally studied this overhead and we noticed that is relative to the architecture of the system. It can be monitored and dynamically adapted based on the load of the system. We also noticed that it can be simply modeled as a constant $\alpha_{bt}$ added to the download time.

**Overhead related to the download phase** In BitTorrent, peers upload to each other even though they may only have parts of the file. This can result in upload interruptions when the uploader has no pieces to offer to his unchoked peers. Fortunately, this problem has already been tackled by Qui *et al.* [100], and the authors proposed a parameter to scale down the upload speed of leechers. This parameter, denoted as $\eta_s \in [0, 1]$, measures the effectiveness of file sharing. It can be computed as follows:

$$\eta_s = 1 - \mathbb{P}\left\{ \begin{array}{c} \text{downloader } i \text{ has no piece that} \\ \text{his unchoked peers need} \end{array} \right\}.$$

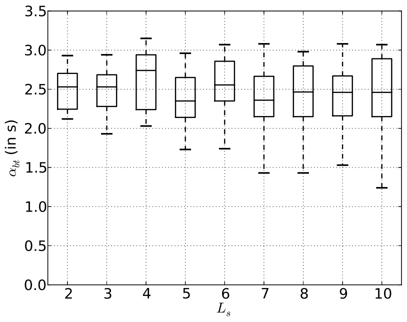
The authors derived this probability and came to the conclusion that $\eta_s$ can be expressed as: [1]

$$\eta_s = 1 - \sum_{n_i=0}^{N_s-1} \frac{1}{N_s} \left( \frac{N_s - n_i}{N_s (n_i + 1)} \right)^{k_s},$$

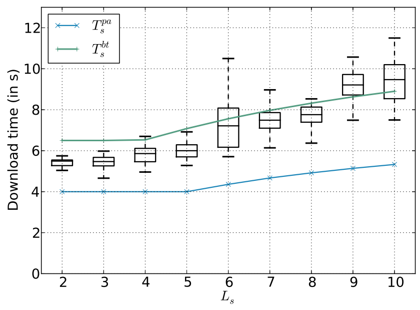
where $N_s$ is the number of pieces of the served file $f_s$, and $k_s$ is the number of connections a peer has.

The authors in [100] focused on the case of large files and concluded that $\eta_s \approx 1$ when $N_s$ is high (which means that the file is big). However, if we consider a small file of 1 MB composed of $N_s = 4$ chunks each of 256 KB, and a peer that has only two connections ($k_s = 2$), the above equation yields $\eta_s = 0.7069$, which means that there is a probability of about 30% that a peer has no pieces for its unchoked peers. This can affect the download time and make it relatively longer. Thus, this overhead should be also considered when estimating the download time in BitTorrent.

[1] The rectified version of [100], which contains the correct expression of $\eta_s$, can be found at: `http://users.encs.concordia.ca/~dongyu/paper/bittorrent.pdf`



(a) $\alpha_{bt}$



(b) $T_s^{bt}$ versus $T_s^{pa}$

**Figure 4.1:** Experimental validation of $T_s^{bt}$

**Download Time with BitTorrent $T_s^{bt}$** Considering the above listed overheads, we can extend (2.3) in order to provide an accurate estimation of the download time in BitTorrent as follows:

$$T_s^{bt} = \frac{F_s}{\min\left\{d_{min,s}, \frac{w_s + \eta_s\, u_s\, L_s}{L_s}, w_s\right\}} + \alpha_{bt}. \tag{4.1}$$

$\eta_s$ is used to scale down the upload capacity of the swarm in order to consider the problem of unavailable pieces. $\alpha_{bt}$, the start-up phase overhead, is added to the download time.

## Experimental Validation

To validate $T_s^{bt}$, we ran repeated experiments using a 1 MB file. The experimental scenario was to distribute the file via BitTorrent starting with a unique seed. The bandwidth setting was the following: the upload bandwidth allocated for the file was $w_s = 5$ Mbps. The number of clients ranged from 2 to 10. Each of them had an upload bandwidth $u_i = 1$ Mbps and a download bandwidth $d_i = 2$ Mbps. In these experiments, we used the standard Vuze[2] client for the BitTorrent seed and clients and a public tracker[3].

[2]Vuze client: http://www.vuze.com/
[3]The OpenBitTorrent tracker: https://openbittorrent.com/

We repeated each experiment 5 times and measured the values of the download time and the overhead $\alpha_{bt}$ for each client. Figure 4.1a represents a box-plot of the time interval between the moments when the clients were launched and when they started downloading the file. This time interval corresponds to $\alpha_{bt}$. We notice that this overhead is constant in average and it is equal to 2.5 seconds for our architecture.

Using this value ($\alpha_{bt} = 2.5$), Figure 4.1b compares our estimation ($T_s^{bt}$), the one proposed in [75] ($T_s^{pa}$), and the experimental results (the box-plots). We can clearly see that $T_s^{pa}$ underestimates the download times with an error rate that can reach 40% in our experiments. On the other hand, our estimation is more accurate with an error rate reduced to about 10%.

### 4.2.3 Gain and Offload Ratios

Sometimes the use of BitTorrent may incur a longer download time compared to HTTP especially when the files are small. The main challenge is to decide which protocol is more suitable for each file transfer. The key elements in making the decision are: (i) the gain in download time which represents the difference in download time between HTTP and BitTorrent, and (ii) the amount of data that can be offloaded from the central server with the use of BitTorrent.

#### The Gain Ratio

To measure the improvement in terms of download time between HTTP and BitTorrent for a given swarm $s$, we introduce the gain ratio $Gain_s$, as follows:

$$Gain_s = \frac{T_s^{http} - T_s^{bt}}{T_s^{http}}.$$

$Gain_s$ represents the ratio between the difference in download times $T_s^{http} - T_s^{bt}$ and $T_s^{http}$. The gain can take different values that can be either negative, positive or equal to zero. A positive gain ratio equal to $x$ means that downloading the file via BitTorrent entails a gain of *100* $x$% in download time compared to HTTP, while a negative value of $Gain_s$ reflects a loss in download time. A gain ratio equal to zero indicates that both protocols (BitTorrent and HTTP) have the same estimated download time. Using (2.2) and (4.1), it is possible to calculate $Gain_s$ based on the divisors of $T_s^{http}$ and $T_s^{bt}$, which are respectively $\min\left\{d_{min,s}, \frac{w_s}{L_s}\right\}$ and $\min\left\{d_{min,s}, \frac{w_s+\eta_s\, L_s\, u_s}{L_s}, w_s\right\}$. We study all the possible cases and we extract the analytic equation of $Gain_s$ as follows:

$$Gain_s = \begin{cases} -\frac{\alpha_{bt}\, d_{min,s}}{F_s}, & \text{if } d_{min,s} \leq \frac{w_s}{L_s} \text{ and } d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, u_s\, L_s}{L_s}, w_s\right\} \\ 1 - \frac{w_s}{L_s\, d_{min,s}} - \frac{\alpha_{bt}\, w_s}{F_s\, L_s}, & \text{if } \frac{w_s}{L_s} \leq d_{min,s} \text{ and } d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, u_s\, L_s}{L_s}, w_s\right\} \\ 1 - \frac{w_s}{w_s+\eta_s\, u_s\, L_s} - \frac{\alpha_{bt}\, w_s}{F_s\, L_s}, & \text{if } \frac{w_s+\eta_s\, u_s\, L_s}{L_s} \leq \min\{d_{min,s}, w_s\} \\ 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, w_s}{F_s\, L_s}, & \text{if } w_s \leq \min\left\{d_{min,s}, \frac{w_s+\eta_s\, u_s\, L_s}{L_s}\right\}. \end{cases} \tag{4.2}$$

*Proof.* We organize the proof into the following four exhaustive cases, based on the values of $\min\left\{d_{min,s}, \frac{w_s}{L_s}\right\}$ and $\min\left\{d_{min,s}, \frac{w_s+\eta_s\, L_s\, u_s}{L_s}, w_s\right\}$.

- If $d_{min,s} \leq \frac{w_s}{L_s}$ and $d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, u_s\, L_s}{L_s}, w_s\right\}$

  $$\begin{cases} d_{min,s} \leq \frac{w_s}{L_s} & \Longrightarrow T_s^{http} = \frac{F_s}{d_{min,s}} \\ d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, u_s\, L_s}{L_s}, w_s\right\} & \Longrightarrow T_s^{bt} = \frac{F_s}{d_{min,s}} + \alpha_{bt} \end{cases}$$

  Thus, $Gain_s = \frac{\frac{F_s}{d_{min,s}} - \frac{F_s}{d_{min,s}} - \alpha_{bt}}{\frac{F_s}{d_{min,s}}} = -\frac{\alpha_{bt}\, d_{min,s}}{F_s}$.

- If $\frac{w_s}{L_s} \leq d_{min,s}$ and $d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, u_s\, L_s}{L_s}, w_s\right\}$

  $$\begin{cases} \frac{w_s}{L_s} \leq d_{min,s} & \Longrightarrow T_s^{http} = \frac{F_s\, L_s}{w_s} \\ d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, u_s\, L_s}{L_s}, w_s\right\} & \Longrightarrow T_s^{bt} = \frac{F_s}{d_{min,s}} + \alpha_{bt} \end{cases}$$

  Thus, $Gain_s = \frac{\frac{F_s\, L_s}{w_s} - \frac{F_s}{d_{min,s}} - \alpha_{bt}}{\frac{F_s\, L_s}{w_s}} = 1 - \frac{w_s}{L_s\, d_{min,s}} - \frac{\alpha_{bt}\, w_s}{F_s\, L_s}$.

- If $\frac{w_s+\eta_s\, u_s\, L_s}{L_s} \leq \min\{d_{min,s}, w_s\}$

  $$\begin{cases} \begin{cases} \frac{w_s}{L_s} \leq \frac{w_s+\eta_s\, u_s\, L_s}{L_s} \\ \frac{w_s+\eta_s\, u_s\, L_s}{L_s} \leq d_{min,s} \end{cases} & \Longrightarrow T_s^{http} = \frac{F_s\, L_s}{w_s} \\ \frac{w_s+\eta_s\, u_s\, L_s}{L_s} \leq \min\{d_{min,s}, w_s\} & \Longrightarrow T_s^{bt} = \frac{F_s\, L_s}{w_s+\eta_s\, u_s\, L_s} + \alpha_{bt} \end{cases}$$

  Thus, $Gain_s = \frac{\frac{F_s\, L_s}{w_s} - \frac{F_s\, L_s}{w_s+\eta_s\, u_s\, L_s} - \alpha_{bt}}{\frac{F_s\, L_s}{w_s}} = 1 - \frac{w_s}{w_s+\eta_s\, u_s\, L_s} - \frac{\alpha_{bt}\, w_s}{F_s\, L_s}$.

- If $w_s \leq \min\left\{d_{min,s}, \frac{w_s+\eta_s\, u_s\, L_s}{L_s}\right\}$

$$\begin{cases} \begin{cases} \frac{w_s}{L_s} \leq w_s \\ w_s \leq d_{min,s} \end{cases} & \Longrightarrow T_s^{http} = \frac{F_s\, L_s}{w_s} \\ w_s \leq \min\left\{d_{min,s}, \frac{w_s+\eta_s\, u_s\, L_s}{L_s}\right\} & \Longrightarrow T_s^{bt} = \frac{F_s}{w_s} + \alpha_{bt} \end{cases}$$

Thus, $Gain_s = \frac{\frac{F_s\, L_s}{w_s} - \frac{F_s}{w_s} - \alpha_{bt}}{\frac{F_s\, L_s}{w_s}} = 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, w_s}{F_s\, L_s}$.

□

## The Offload Ratio

The offload ratio defines the amount of data offloaded from the central content provider. It is determined by the total amount of data exchanged between the peers divided by the total volume of downloaded data:

$$\begin{aligned} \mathit{Offload}_s &= \frac{\mathit{data\ from\ peers}}{\mathit{total\ data\ sent}} = 1 - \frac{\mathit{data\ from\ the\ central\ server}}{\mathit{total\ data\ sent}} \\ &= 1 - \frac{\sum\limits_{c_i \in S} \int_{\alpha_{bt}}^{T_s^{bt}} r_i(t)dt}{F_s\, L_s} = 1 - \frac{\left(T_s^{bt} - \alpha_{bt}\right) \sum\limits_{c_i \in S} r_i(t)}{F_s\, L_s}, \end{aligned}$$

where $r_i(t)$ is the seeding rate as described in Chapter 2.

Taking into consideration the seeding rate as defined in (2.4), we can formulate the offload rates as follows:

$$\mathit{Offload}_s = \begin{cases} 1 - \frac{1}{L_s} & \text{if } d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, L_s\, u_s}{L_s}, w_s\right\} \text{ and } d_{min,s} \leq \frac{u_s\, L_s}{L_s-1} \\ \frac{\eta_s\, u_s}{d_{min,s}} & \text{if } d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, L_s\, u_s}{L_s}, w_s\right\} \text{ and } \frac{u_s\, L_s}{L_s-1} \leq d_{min,s} \\ 1 - \frac{w_s}{w_s+\eta_s\, L_s\, u_s} & \text{if } \frac{w_s+\eta_s\, L_s\, u_s}{L_s} \leq \min\left\{d_{min,s}, w_s\right\} \\ 1 - \frac{1}{L_s} & \text{if } w_s \leq \min\left\{d_{min,s}, \frac{w_s+\eta_s\, L_s\, u_s}{L_s}\right\} \end{cases} \tag{4.3}$$

*Proof.* The proof of $\mathit{Offload}_s$ is organized into the same cases as $r_i(t)$, as follows:

- If $d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, L_s\, u_s}{L_s}, w_s\right\}$ and $d_{min,s} \leq \frac{u_s\, L_s}{L_s-1}$

$$\sum_{c_i \in S} r_i(t) = \sum_{c_i \in S} \frac{u_i\, d_{min,s}}{\eta_s\, u_s\, L_s} = \frac{d_{min,s} \sum\limits_{c_i \in S} u_i}{\eta_s\, u_s\, L_s} = \frac{d_{min,s}\, \eta_s\, L_s\, u_s}{\eta_s\, u_s\, L_s} = d_{min,s}$$

Thus, $\mathit{Offload}_s = 1 - \frac{\frac{F_s}{d_{min,s}}\, d_{min,s}}{F_s\, L_s} = 1 - \frac{1}{L_s}$

- If $d_{min,s} \leq \min\left\{\frac{w_s+\eta_s\, L_s\, u_s}{L_s}, w_s\right\}$ and $d_{min,s} \geq \frac{u_s\, L_s}{L_s-1}$

$$\begin{aligned}\sum_{c_i \in S} r_i(t) &= \sum_{c_i \in S} \left(\frac{u_i-\eta_s\, u_s\, L_s}{L_s-1} + d_{min,s}\right) = \frac{\sum\limits_{s \in S}(u_i-\eta_s L_s u_s)}{L_s-1} + L_s\, d_{min,s} \\ &= \frac{\eta_s\, L_s\, u_s-\eta_s\, L_s^2\, u_s}{L_s-1} + L_s\, d_{min,s} = L_s(d_{min,s} - \eta_s\, u_s)\end{aligned}$$

Thus, $\mathit{Offload}_s = 1 - \frac{\frac{F_s}{d_{min,s}}\, L_s(d_{min,s}-\eta_s\, u_s)}{F_s\, L_s} = \frac{\eta_s\, u_s}{d_{min,s}}$

- If $\frac{w_s+\eta_s\, L_s\, u_s}{L_s} \leq \min\{d_{min,s}, w_s\}$

$$\begin{aligned}\sum_{c_i \in S} r_i(t) &= \sum_{c_i \in S} \left(\frac{u_i-\eta_s\, u_s\, L_s}{L_s-1} + \frac{w_s+\eta_s\, u_s\, L_s}{L_s}\right) \\ &= \frac{\sum\limits_{s \in S}(u_i-\eta_s L_s u_s)}{L_s-1} + w_s + \eta_s\, u_s\, L_s \\ &= w_s + \eta_s\, u_s\, L_s - \frac{L_s^2\, \eta_s\, u_s - L_s\, \eta_s\, u_s}{L_s-1} = w_s\end{aligned}$$

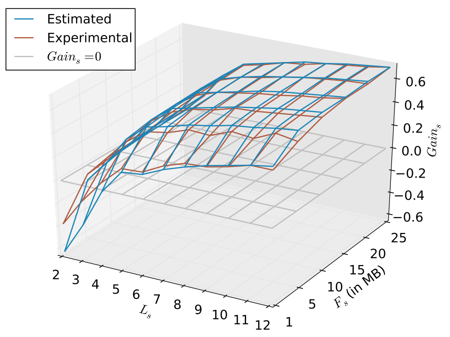
Thus, $\mathit{Offload}_s = 1 - \frac{\frac{L_s\, F_s}{w_s+\eta_s\, u_s\, L_s}\, w_s}{F_s\, L_s} = \frac{w_s}{w_s+\eta_s\, u_s\, L_s}$

- If $w_s \leq \min\left\{d_{min,s}, \frac{w_s+\eta_s\, L_s\, u_s}{L_s}\right\}$

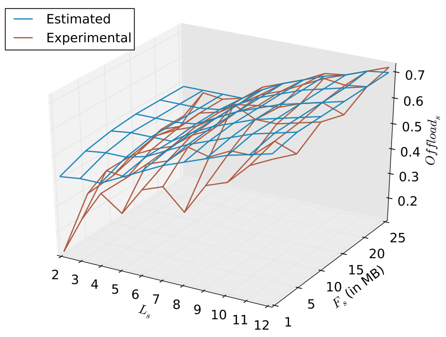
$$\sum_{c_i \in S} r_i(t) = \sum_{c_i \in S} \frac{u_i\, w_s}{\eta_s\, u_s\, L_s} = \frac{\eta_s\, L_s\, u_s\, w_s}{\eta_s\, u_s\, L_s} = w_s$$

Thus, $\mathit{Offload}_s = 1 - \frac{\frac{F_s}{w_s}\, w_s}{F_s\, L_s} = 1 - \frac{1}{L_s}$

$\square$



**(a)** *$Gain_s$*



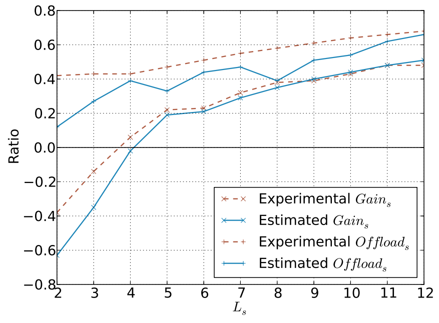
**(b)** *$Offload_s$*

**Figure 4.2:** Estimated versus experimental gain and offload ratios. The swarms' sizes range from 2 to 12 clients and the files from 1 to 25 MB.

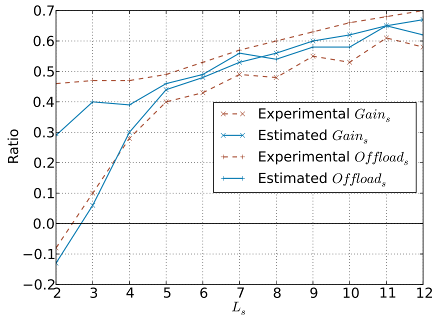
## Experimental Validation

Since the gain and the offload are key parameters in the protocol decision process, it is important to verify the accuracy of our estimation compared to real experimental values. We run experiments using the same bandwidth distribution as in the previous section. The goal is to compare the experimental and the estimated gain and offload ratios when distributing a file to a set of nodes whose number ranges from 2 to 12 nodes. The file size varies from 1 to 25 MB.

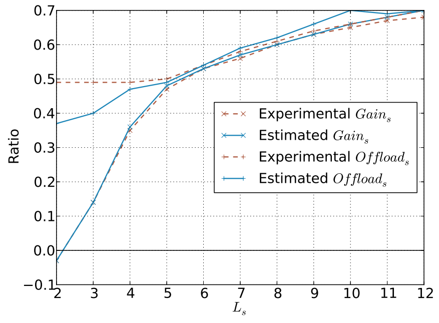
Figures 4.2a and 4.2b represent 3-dimensional plots of the results. We can see that the experimental and estimated surfaces of the $Gain_s$ are very close and that the difference between them is slight. This proves that (4.2) is quite accurate. In fact, the median error rate is about 2.41% (the mean error is equal to 5.22%). The figure also shows that the gain in download time $Gain_s$ tend to increase when the swarm and file sizes increase. This means that BitTorrent performs better with big files/swarms, but it does not deny that it can perform well with the small ones too. The $Gain_s = 0$ plane, which corresponds to a similar performance between HTTP and BitTorrent ($T_s^{http} = T_s^{bt}$), separates both the estimated and experimental surfaces into 2 parts. The parts of the plots above this plane reflect that BitTorrent outperforms HTTP ($T_s^{bt} < T_s^{http}$), while the parts below reflect degrades in performance using BitTorrent ($T_s^{bt} > T_s^{http}$). By looking at



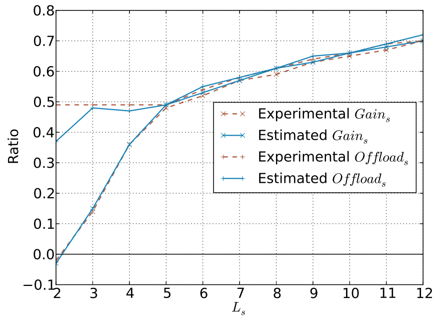
**(a)** 1 MB



**(b)** 5 MB



**(c)** 20 MB



**(d)** 25 MB

**Figure 4.3:** Estimated versus experimental gain and offload ratios for different file sizes

Figure 4.2a, we can clearly see that most of the plot is above the plane which means that in most cases, BitTorrent outperforms HTTP.

Figure 4.2b compares the experimental and the estimated *Offload*$_s$. The offload is always positive since the use of BitTorrent involves some cooperation between the clients. However, the median error rate is equal to 3.47% (the mean error is equal to 14.13%). This is mainly due to the unpredictable nature of the BitTorrent protocol. Nevertheless, this error can still be considered among the accepted margin of errors.

Figures 4.3a, 4.3b, 4.3c and 4.3d present some vertical sections of the previous plots for files of size $1, 5, 20$ and $25$ megabytes, respectively. The figures contain the corresponding estimated and experimental values of both the gain and the offload ratios. We notice that the estimations are very close to the experimental results in most cases. For instance, for the smallest file (1 MB), the error in the gain estimation is moderate for very small swarms with only 2 or 3 clients. That error could represent an increase in the download time of a few seconds. However, we notice that the bigger the swarms is, the closer the estimation gets, in a way that the error becomes very close to 0 for swarms of size $\geq 4$ clients.

## 4.3 The Protocol Decision Strategy

In this section, we present a simple methodology that data centers can deploy in order to reduce the load on their servers. The main idea is to switch the download protocol from HTTP to BitTorrent when there are simultaneous downloads of the same file. We start by presenting the download scenario and explain how can BitTorrent be introduced to offload the central servers. After that, we propose four criteria that define the switching point from HTTP to BitTorrent. Finally, we present the switching algorithm which formalizes the protocol decision strategy into step-by-step operations.

### 4.3.1 Download Scenario

The download requests that arrive to the data center are HTTP GET requests sent from different clients. Each request is issued from one client $c$ and translates its interest in downloading one file $f$. Using the traditional model, this request will be fulfilled by the data center and the file $f$ will be sent directly to $c$ via HTTP.

With our strategy, before sending the file, the data center verifies if $f$ is already being downloaded by other clients. If it is not the case, then the file is sent via HTTP as usual. Otherwise, if there are already some other clients downloading $f$, the data center decides if it is worth switching to BitTorrent for the corresponding file. This decision is made based on the current load on the data center and the characteristics of the clients downloading the file.

If the data center decides that BitTorrent is not efficient for the swarm, the file is sent via HTTP. Otherwise, the download protocol is switched from HTTP to BitTorrent for all he peers requesting $f$. All these steps

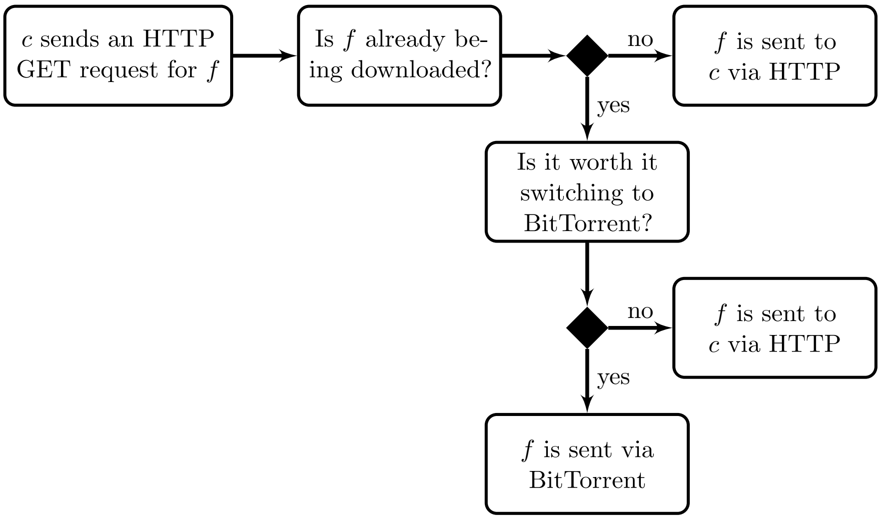

$c$ sends an HTTP GET request for $f$
Is $f$ already being downloaded?
no
$f$ is sent to $c$ via HTTP
yes
Is it worth it switching to BitTorrent?
no
$f$ is sent to $c$ via HTTP
yes
$f$ is sent via BitTorrent

**Figure 4.4:** File download scenario

are summarized in Figure 4.4.

### 4.3.2 Switching Constraints

We presented in the previous section two key parameters that can help in measuring the trade-offs between HTTP and BitTorrent (i) the gain ratio which quantifies the gain or loss in time that the leechers might experience when switching from HTTP to BitTorrent, and (ii) the offload ratio which gives an estimation of the amount of data that can be offloaded from the server thanks to BitTorrent.

We remind the reader that the main reason behind the use of BitTorrent is to offload the data center from doing all the serving. However, the use BitTorrent can have a negative effect of the QoS. It is clear that if we neglect a potential increase in download time caused by the use of BitTorrent, the overall offload ratio will always be the highest possible. However, it is equally important for the data center to maintain a significant level of QoS. Thus, it is important to study the trade-off between the offload ratio and the gain in terms of download times. To do so, the data center can put different constrains on the parameters. To this extent, we distinguish the four following cases based on the constraints that can be placed on these parameters:

1. The first possible solution is to put no constraints, that is, BitTorrent is always used when the number of leechers $L_s \geq 2$. In this case, the overall offload ratio will be the highest possible. But, the clients might experience an excessively longer download time.

2. Another possible solution is to put a limit on the offload ratio: the data center switches to BitTorrent only when the offload is important. For example, the data center can decide to switch only when the estimated offloaded bandwidth is above 50% of the total bandwidth, regardless of the download time.

3. The third possible case is to fix a gain limit: the data center decides to switch only when the download time in BitTorrent compared to HTTP does not exceed a certain threshold. This threshold can be put on the gain ratio to ensure a minimal bound on the permitted loss in download time.

4. The last possibility is to fix both the gain and offload ratios. While this case presents an efficient strategy to be selective, it might be too strict and could limit the overall offload ratio.

The system administrator is the sole responsible of fixing the switching constraints based on the system's requirements. The manager has to take into account that putting no limits is not a wise decision as it can degrade significantly the download times, a limit on the offload is beneficial for the data center, but does not benefit the clients, and fixing both limits can be too strict and reduce the probability of switching. To this extent, we consider in this thesis a switching strategy based on the third case. We pose $\tau$ as a gain constraint that satisfies the condition $Gain_s \geq \tau$. If $\tau \leq 0$, it means that the system tolerates a potential increase in the download time that could occur because of the switch. However, a positive value of $\tau$ reflects a stricter constraint. For instance, $\tau = -0.5$ means that an increase up to 50% of the download time is tolerated. Note that a constraint of this magnitude is possible, because $\tau = -0.5$ could represent, for small files, a slight increase in the download time, in the order of a few seconds, to be more precise.

$\tau$ can take different values depending on the services offered by the data center. The choice of its concrete value is left up to the system administrator. A possible concrete example of $\tau$ is the following: Suppose that a given service provider cannot gain in bandwidth at the expense of worsening the download time for premium users who are paying money for the service. For this type of clients, $\tau$ should be always $\geq 0$.

### 4.3.3 Switching Algorithm

For an easy management of the download protocols, we propose Algorithm 1, which is executed for each swarm $s$ upon the arrival of each new download request on $f_s$. We suppose that our system keeps track of the state of each swarm $s$ as a boolean value $switched_s$, where $switched_s$= *True* if the current download protocol is BitTorrent (the switch has already taken place). Otherwise, $switched_s$=*False*.

The algorithm works as follows: First, the system verifies if the file is already being downloaded by other clients, beside the new requester (*line 1*). If it is not the case (the new requester is the only requester of $f_s$), the file is sent via HTTP and the algorithm is terminated (*line 23*).
In the case where there are already some clients downloading $f_s$, the system verifies the download protocol already in use to distribute $f_s$ (*line 2*). If $f_s$ is being downloaded by the default protocol (HTTP), the system computes the estimated gain and compares it with $\tau$ (*lines 3 and 4*). If the resulting gain is below the constraint, the file will be sent to the requester via HTTP (*line 14 to 16*). Otherwise, the distribution protocol will be switched to BitTorrent (*lines 5 to 12*). A `.torrent` file will be created and sent to all the clients requesting $f_s$.

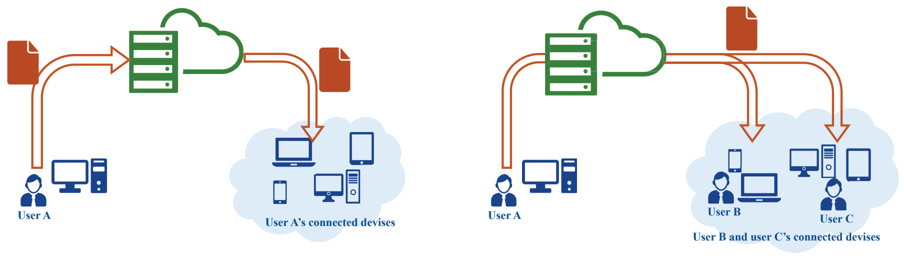
In parallel, a seed will be launched in the data center. Upon the reception of the `.torrent` file, a BitTorrent leecher will be launched inside each of these clients and they will start downloading the file in BitTorrent, while offloading the data center from doing all the serving.

If $f_s$ is already being distributed by BitTorrent, then the `.torrent` will be send to the new requester who can directly join the existing swarm and start downloading $f_s$ via BitTorrent (*lines 19 and 20*).

**Algorithm 1** Switching Algorithm

```
Input $\tau$                    ▷ the switching constraint
Input $switched_s$              ▷ the state of the swarm $s$
Input $F_s$                     ▷ the size of file $f_s$
Input $w_s$      ▷ the data center's upload bandwidth allocated to $s$

1: if there are other clients downloading $f_s$ then
2:     if not $switched_s$ then
3:         calculate $Gain_s$
4:         if ($Gain_s \geq \tau$) then
5:             create a .torrent related to $f_s$
6:             launch a BitTorrent seed in the data center
7:             for each client $c_i$ in $s$ do
8:                 $c_i$ downloads the .torrent from the server
9:                 $c_i$ launches a BitTorrent leecher
10:                $c_i$ starts BitTorrent transfer
11:            end for
12:            $switched_s$= True
13:        else
14:            for each client $c_i$ in $s$ do
15:                $c_i$ downloads $f_s$ via HTTP
16:            end for
17:        end if
18:    else
19:        send the .torrent to the new requester
20:        launch a BitTorrent leecher inside that requester
21:    end if
22: else
23:    send $f_s$ via HTTP
24: end if
```


User A
User A's connected devises
User A
User B
User C
User B and user C's connected devises

(a) Synchronization (b) Sharing

**Figure 4.5:** Synchronization and sharing in Personal Clouds

## 4.4 Application Scenario: Personal Clouds

In this section, we present Personal Clouds as a possible application scenario of the switching algorithm. We start by describing two scenarios where BitTorrent can be introduced in Personal Clouds. Next, we present the necessary elements needed to extend classic Personal Clouds with BitTorrent features. After that, we outline the possible security concerns and present the measures to solve them. Finally, we investigate the effects of the switching algorithm on the U1 trace, previously described in Chapter 2, and we prove that the use of BitTorrent can reduce significantly the bandwidth cost on the Personal Cloud provider.

### 4.4.1 Sharing and Synchronization

The protocol switching approach can be applied widely in any cloud-based system. Personal Clouds are the most appropriate for this proposal since the developers can tune the client's implementation to extend it with the BitTorrent functionalities. The two following common file distribution scenarios could benefit from the hybrid download strategy:

- **Synchronization:** One of the key aspects of a Personal Cloud is file synchronization (called also syncing) which ensures that all the devices of a given user contain the same, up-to-date, files. This means that every copy of a file should be identical in all locations. In Personal Clouds, file synchronization is bi-directional. This means that a locally modified file is updated in each location where this file is present. Similarly, if a file is modified remotely, the change will be automatically updated locally [52]. Figure 4.5a presents an example of file synchronization. User A is adding a new file $f_s$ to his personal account. During the synchronization process, the same file will be download by all the other synchronized devices of the user. These devices represent the elements of the swarm $s$.

- **Sharing:** Sharing is an attractive feature that most Personal Clouds provide. Sharing can be internal, between users inside the service, or public, involving people from outside the Personal Cloud. Internal sharing is usually offered as an integrated functionality in the user interface, whereas public sharing is commonly offered as direct HTTP links that allow other users to access to certain files or folders [52]. Figure 4.5b presents an example of file sharing. User A is sharing a file $f_s$ with user B and user C. In this case, the file will be downloaded by all the synchronized devices of the users. All the connected devices of User B and User C form the swarm $s$.

Synchronization and sharing are quite common in Personal Clouds. In [37], the authors analyzed a dataset based on the behavior of read Dropbox users . They noticed that about 40% of the households had more than one device linked to the service and that most of these households had up to 4 devices connected. Moreover, about 60% of these households with more than one device have at least one shared folder. In the U1 trace described in Chapter 2, we found that more than 11% of the files were downloaded more than once which corresponds to more than 30% of the measured download operations.

## 4.4.2 Extending Personal Clouds with BitTorrent

As an application example, we consider a classic Personal Cloud (Store, Sync and Share functionalities), enriched with extra components that allow inter-client content transfers via BitTorrent. The architecture of the system is presented in Figure 4.6. Compared to the typical architecture of a Personal Cloud (Figure 2.5), several components are added to accommodate the BitTorrent behavior, including:

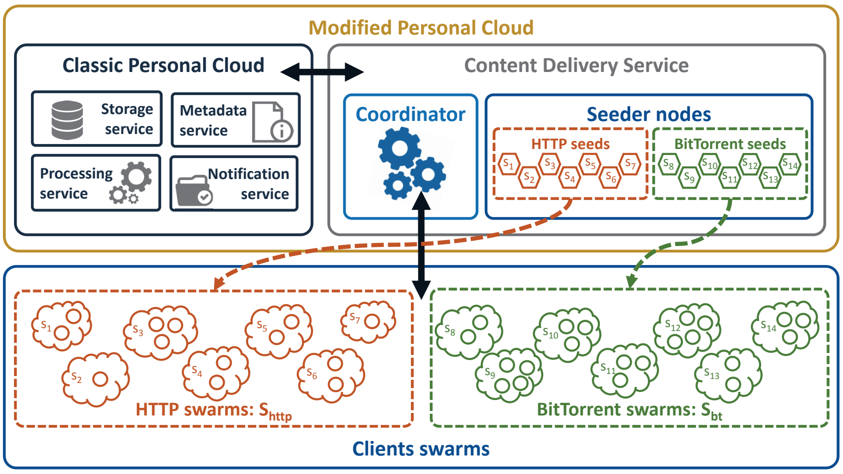


**Figure 4.6:** Global view of the system architecture

- **Content Delivery Service :** The content delivery service is responsible for processing the requests coming from the end-users and ensuring the delivery of the files to the corresponding requesters. The main components added, compared to the default architecture (Figure 2.5), are:

  – **Coordinator:** The coordinator is the core component of the Personal Cloud. It is responsible for managing all the clients' requests and ensuring they are processed correctly. The coordinator is also responsible for the proper management of the Personal Cloud's resources.

  – **Seeder nodes:** The seeder nodes are the entities responsible for delivering the requested content from the storage back-end servers to the end-users. To each file being distributed corresponds one seeder node. In this thesis, we refer to these seeder nodes as *cloud seeds* or *seeds*. We distinguish two types of seeds: *HTTP seeds* and *BitTorrent seeds* depending on the algorithm adopted to distribute the corresponding requested content to end-users.

- **Clients swarms:** All the end-user peers are organized into swarms. We define a swarm by the set of peers that are requesting the same file. If a file is being downloaded by a single peer, we consider it as a single-peer swarm. This means that, at a given time, there are as many swarms as the number of files being downloaded (to each file corresponds only one swarm and one seeder node). In our model, we distinguish between two types of swarms:
  - **HTTP Swarms:** The HTTP swarms are the swarms whose peers are downloading the corresponding file via HTTP. Clearly, these peers are not collaborating with each other, but grouping them in swarms is a simple means of control which will help, later on, in making the switching decision.
  - **BitTorrent Swarms:** Similar to HTTP swarms, BitTorrent swarms are the swarms whose peers are downloading the corresponding file from BitTorrent seeds via the BitTorrent protocol. Typically, these swarms are composed of two peers or more. Since the peers are supposed to collaborate between each other with the help of the cloud seeds, it makes no sense to have a single-peer BitTorrent swarm.

In our Personal Cloud system, all the users' files are uploaded using HTTP over an encryption layer of SSL/TLS. To download a file, the client sends an HTTP GET request to the coordinator. The latter verifies the existence of the file in the storage nodes and decides the download protocol to be used: HTTP or BitTorrent. The decision is made based on the load on the seed and the swarms' characteristics, as detailed in Algorithm 1. In the case of an HTTP download, an HTTP seed is associated with the requested file, which will be transferred using HTTP (over SSL/TLS). Otherwise, if the coordinator decides to switch to BitTorrent, a `.torrent` meta-data file is created and a corresponding BitTorrent seed in launched. After that, the recently created `.torrent` file will be transmitted to the corresponding clients. These clients, unaware of all these interactions, will then start downloading the file using the BitTorrent protocol (from the cloud seeds and/or from the other clients). Evidently, the "old" clients who arrived before the switch to BitTorrent will also benefit from the switch if they did not finish the download. In fact, when an "old" client requests

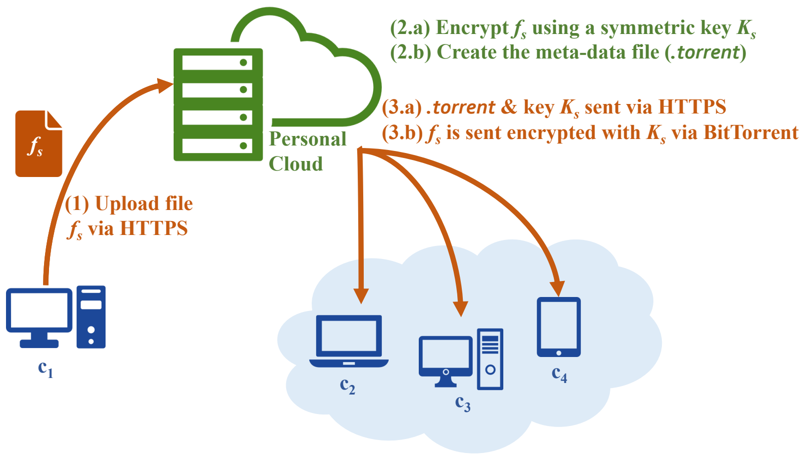

(2.a) Encrypt $f_s$ using a symmetric key $K_s$
(2.b) Create the meta-data file (*.torrent*)
(3.a) *.torrent* & key $K_s$ sent via HTTPS
(3.b) $f_s$ is sent encrypted with $K_s$ via BitTorrent
$f_s$
Personal Cloud
(1) Upload file $f_s$ via HTTPS
$c_1$
$c_2$
$c_3$
$c_4$

**Figure 4.7:** Encryption mechanism with BitTorrent: Before being sent to clients $c_2$, $c_3$ and $c_4$ via BitTorrent, the file $f_s$ is encrypted on the server's side using a one-shot symmetric key $K_s$. The `.torrent` meta-data file is sent, along with the encryption key $K_s$ via HTTPS.

a new part of the file to be downloaded, he will realize that the transfer protocol has changed and will automatically adapt to the new one. Thus, each "old" client will join the swarm with the pieces he already has, which means that he will be probably contributing to the swarm as soon as he switches to BitTorrent in a very transparent way.

**Security Concerns** In most Personal Clouds, all the client-server communication is over HTTPS, which protects against eavesdropping and tampering with the contents of the communication. To ensure data confidentiality when the transfer protocol is BitTorrent, we deploy a one-shot symmetric encryption mechanism that uses unique keys to encrypt the files that will be transferred via BitTorrent. In fact, when the decision of using BitTorrent for a given file $f_s$ is made, the cloud creates a corresponding meta-data `.torrent` file, and assigns to $f_s$ a unique one-shot symmetric encryption key $K_s$ (such as a DES or AES key). This key, along with the `.torrent` file, are sent to each of the requesters via HTTPS. The file $f_s$ is encrypted by the server using $K_s$ before being sent to the requesters via BitTorrent. More details are provided in Figure 4.7. When a client

**Table 4.3:** Offloaded volume and offload percentages resulting from the application of Algorithm 1 using different $\tau$ values.

| Constraint | Offloaded Volume | Overall Offload % |
|---|---|---|
| $\tau = -1.0$ | 207.35 GB | 16.7183% |
| $\tau = -0.5$ | 207.33 GB | 16.7170% |
| $\tau = -0.2$ | 207.04 GB | 16.6938% |
| $\tau = 0.0$ | 137.64 GB | 11.0979% |
| $\tau = 0.2$ | 137.59 GB | 11.0942% |
| $\tau = 0.5$ | 90.60 GB | 7.3055% |
| $\tau = 1.0$ | 0.0 GB | 0.0% |

completes the download, it simply decrypts the protected file with the received key and the synchronization process terminates. Note that integrity is already guaranteed by BitTorrent itself, so no additional hash computations are needed.

### 4.4.3 Implication of the Switching Algorithm

We applied Algorithm 1 on the trace using the following settings: (i) the upload speed of the seed allocated to each swarm is $w_s = 2$ Mbps[4], (ii) the clients are homogeneous and have an upload and download speed of 512 Kbps and 1 Mbps, respectively, and (iii) the peers discovery overhead is $\alpha_{bt} = 2.5$ seconds.

We went through the trace focusing on the files that have been downloaded more than once in order to identify the files with collapsing download times which are the candidates for the switch to BitTorrent. In other words, for each file, we checked if there were consecutive download operations (at time stamps $t_1$ and $t_2$) that came before the end of the theoretical download time in HTTP: that is, $t_2 - t_1 \leq T_s^{http}$. The download time $T_s^{http}$ is calculated using (2.2), based on the settings listed above. After the identification of these files, we calculated for each case the gain ratio using (4.2). Depending on the gain value and the $\tau$ constraint, we identified the files that were subject to switching and measured the corresponding offloaded volume of data using (4.3).

[4]We remind the readers that $w_s$ does not refer to the total upload bandwidth of the cloud, but to the portion of its bandwidth allocated to the each specific file/swarm

**Table 4.4:** Data transfer pricing from Amazon S3 to Internet in the European region.

| Data transfer interval | Pricing |
|---|---|
| The first 1 GB/month | \$0.0 per GB |
| Up to 10 TB/month | \$0.12 per GB |
| Next 40 TB/month | \$0.09 per GB |

Table 4.3 presents the results of the application of Algorithm 1 on the trace. The overall offload percentage is calculated based on the percentage ratio between the offloaded volume and the total downloaded volume for the whole trace (1,240.25 GB). We vary the values of the switching constraint $\tau$ in order to get a global idea of the gains. We notice that if we fix $\tau$ to tolerate losses of 20% ($\tau = -0.2$), the cloud load can be reduced up to 16%. In the case of stricter constraints: no loss is tolerated ($\tau = 0$), or no switch unless there is a 20% gain in download time ($\tau = 0.2$), the overall offload percentage falls down to around 11%. We notice that the stricter the constraint is, the lower the overall offload is. This is mainly because stricter constraints reduce the probability of switching to BitTorrent, which leads to limited offload. A very strict constraint $\tau = 1.0$ can never be reached ($tau = 1.0 \Rightarrow Gain_s \geq 1 \Rightarrow T_s^{bt} \leq 0$). Thus, no swarms would be switching to BitTorrent, which explains why the corresponding overall offload percentage is equal to 0.

Even though the U1 system is not very popular, our algorithm could achieve savings up to 16% in terms of cloud bandwidth. We strongly believe that this offload would be higher on other systems, like Dropbox or Google Drive, which involve more users and more file sharing.

**Monetary Cost** To measure the amount of money that can be saved using our algorithm, we consider a Personal Cloud that uses Amazon Simple Storage Service (S3) as a storage back-end (like U1). Table 4.4 presents the standard charging rates for data transfer at the time of writing this thesis[5]. Using these rates, and fixing the gain constraint to $\tau = -1$, our switching strategy would lead to savings of about 15% of the overall data transfer cost. We believe that these savings can be even higher for systems that involve more sharing than U1.

[5]More information about the complete and updated rates can be found at http://aws.amazon.com/s3/pricing/

**Table 4.5:** Results using file grouping

| Grouping Period | Constraint | Offloaded Volume | Overall Offload % |
|---|---|---|---|
| **10 seconds** | $\tau = -0.2$ | 213.97 GB | 17.2526% |
| | $\tau = 0.0$ | 140.95 GB | 11.3658 % |
| **30 seconds** | $\tau = -0.2$ | 214.43 GB | 17.2895 % |
| | $\tau = 0.0$ | 140.95 GB | 11.3658 % |

**Effect of file bundling** Bundling consists in grouping a batch of small files that need to be transferred as a single object. This technique is used by Dropbox [36] in an attempt to reduce both transmission latency and control overhead.

If we take a look at (4.2), we notice that $Gain_s$ and the file size $F_s$ are related in a way that if $F_s$ increases, $Gain_s$ will increase too. Similarly, file bundling should presumably increase the overall offload, too. Here, we study the effect of applying this technique in our trace. For a given "bundling period", we group the files that are requested by the same users and consider them as a single file, so that a single `.torrent` file is created for all of them.

Table 4.5 shows the results of grouping files considering 2 different bundling periods. We notice that, compared to the results presented in Table 4.3, bundling is not very effective in this scenario: a slight improvement of the overall offload percentage in the order of 0.55% for $\tau = -0.2$ and about 0.26% for $\tau = 0$. Even with a long grouping period of 30 seconds, the increase of the overall offload percentage remains limited: in the order of 0.03% compared to a bundling period of 10 seconds. However, these results do not imply that the use of this technique could not be effective in increasing the offload rate in other systems.

## 4.5 Conclusions

In this chapter, we presented two metrics that can be used to measure the efficacy of HTTP and BitTorrent for file transfer: the gain and the offload ratios. They can be used in many scenarios as a means of evaluation of the most appropriate download protocol for each case. We used these metrics to develop a solution for reducing costs in Personal Clouds. The cloud server can benefit from the spare upload bandwidth of the clients by switching the transfer protocol from HTTP to BitTorrent. Our proposed algorithm for the management of download protocols studies for each specific case the benefits that can be derived from the switch. Based on the threshold fixed on download time, the algorithm predicts the best protocol to use. Our proposal is validated using a real trace of a Personal Cloud system.

Even though, we considered the worst case scenario, when clients leave the system as soon as they finish the download, the results show that about 16% of the provisioned cloud bandwidth can be saved without degrading the download time service. However, in Personal Clouds, normally peers stay for a period of time after the synchronization process is over. This means that they will be more present and will contribute even more. Thus, we believe that the cloud gains will be even more important in this case.

# 5 Bandwidth Allocation Strategy

In this chapter, we propose a bandwidth allocation strategy for data centers in order to manage their available bandwidth among the different clients. First, we state the bandwidth allocation problem that can occur when data centers have to server both HTTP and BitTorrent clients simultaneously. Next, we study the relationship between the amount of bandwidth allocated to a swarm of clients and the resulting download time. Based on a fixed QoS constraint, we calculate the amount of data center bandwidth needed to ensure a given ratio between the download times in HTTP and BitTorrent. After that, we propose a dynamic algorithm that decides the most suitable protocol for each case and provides the corresponding bandwidth allocations at the swarm level. The algorithm is evaluated later using the U1 trace and the results show important improvements in the download time experienced by the peers, despite the limited bandwidth budget.

*The results presented in this chapter are published in [21]*

## 5.1 Introduction

We consider in this chapter a data center that can deliver files to the end-users using both protocols HTTP and BitTorrent. This means that at the same time, the data center is required to serve both kind of clients: the ones that use HTTP and the ones that use BitTorrent. In these systems, HTTP is the default protocol used and BitTorrent can be introduced when there are simultaneous requests from different clients for the same file. We proposed in the previous chapter, a simple protocol decision strategy that consists in calculating for each file being downloaded by more than one client, how much these clients would gain in terms of download time if they used BitTorrent instead of HTTP. If that gain satisfies the predefined Quality of Service (QoS) constraint, then BitTorrent is chosen. Otherwise, HTTP is kept as the download protocol.

The switching decision is made based on the assumption that the data center allocates the same amount of bandwidth to each file being served. This means that when the number of files to download increases, the data center is required to increase its total bandwidth output. Even though this approach is simple and direct, it is not convenient for data centers that have bandwidth budget constraints. When the data center's upload bandwidth is limited, the different swarms will have to compete over this bandwidth and it is essential to set some rules to define the bandwidth allocation strategies. This chapter deals with this bandwidth allocation problem. It is dedicated to answering the second question addressed in the introductory chapter which is: *How to balance the available bandwidth resources between the concurrent HTTP and BitTorrent swarms?* To the best of our knowledge, there is no related work that deals with both kind of protocols. But, there have been some research works that addressed the bandwidth allocation in HTTP servers and peer-assisted systems based on BitTorrent separately.

In this chapter, we propose a bandwidth allocation strategy that data centers can adopt in order to manage their available bandwidth among the different clients. First, we state the bandwidth allocation problem in Section 5.2. Next, we study the relationship between the amount of bandwidth allocated to a swarm of clients and the resulting download time in Section 5.3. Based on a fixed QoS constraint, we calculate the amount of data center bandwidth needed to ensure a given ratio between the download times in HTTP and BitTorrent. In Section 5.4, we propose a dynamic algorithm which uses simple parameters that can be collected by the system and evaluates the efficacy of using HTTP and BitTorrent as a distribution protocol for each requested file. Based on the load on the data center and the predefined switching constraints, the algorithm decides the most suitable protocol for each case and provides the corresponding bandwidth allocations at the swarm level. Section 5.5 evaluates the efficiency of our proposal using the U1 trace: We vary the QoS constraints and the data center's bandwidth limits and measure the degree of improvement in download time of the involved clients using our algorithm (BitTorrent and HTTP together) compared to the use of HTTP alone. The results show important improvements in the download time experienced by the peers, despite the limited bandwidth budget. Finally, Section 5.6 discusses the results and concludes the chapter.

## 5.2 Bandwidth Allocation Problem

We consider a data center that has a maximum upload bandwidth equal to $W$. This bandwidth is to be shared by the different concurrent swarms in the system. These swarms can be either *HTTP swarms* or *BitTorrent swarms*. An HTTP swarm is a group of clients downloading the same file from the data center using HTTP, while a BitTorrent swarm is a collection of clients downloading a given file via BitTorrent.

To each swarm $s$, the data center allocates a part $w_s$ of its upload bandwidth $W$. These bandwidth allocations $w_s$ are measured in bandwidth unit. They should be non-negative ($w_s \geq 0, \forall s \in S$) and their sum should not exceed the data center's upload budget limit ($\sum\limits_{s \in S} w_s \leq W$).

The bandwidth allocation strategy follows these two rules:

**Rule 1.** *HTTP is the default download protocol and each swarm is allocated a share of the data center's bandwidth equal to its demand.*

**Rule 2.** *The data center can decide to switch the download protocol for a given swarm $s$ from HTTP to BitTorrent if it (the data center) will save in bandwidth and if this change of protocol will not jeopardize the QoS constraint related to the download time of the clients in $s$.*

Rule 1 defines HTTP as the default download protocol and sets the share of the data center's bandwidth allocated to $s$ to be equal to the aggregated download speeds of the peers in $s$: $w_s = D_s$, where $D_s = \sum_{c_i \in s} d_i$ is the sum of the download speeds of all the peers in $s$. When it is possible to gain in bandwidth, the data center can decide to switch the download protocol from HTTP to BitTorrent, as stated in Rule 2. This change of protocols is fixed by a QoS constraint $Gain_s \geq \tau$. This constraint defines the degree of improvement (or degrade) in download time that is accepted when considering the switch to BitTorrent. For instance, $\tau = 0.2$ requires an improvement of a least 20% in download time if BitTorrent is used instead of HTTP. A negative value of $\tau$ reflects degrades in download times. $\tau = -0.5$ means that losses in download time up to 50% are accepted. The bandwidth allocated to BitTorrent swarms should be the minimum that satisfies the switching constraint, which represents the solution of $Gain_s = \tau$. As the number and size of swarms evolve over time with new peers joining and leaving, we will need to adapt the assignments and allocations accordingly since the data center's upload speed $W$ is limited.

In order to satisfy Rule 2, the data center should be able to determine the exact amount of bandwidth needed to satisfy the QoS constraint before switching to BitTorrent. This means that it should be able to estimate the amount of bandwidth $w_s^*$ that can ensure the limit $Gain_s = \tau$. This amount is presented in the following section along with a complete proof in Appendix A.

## 5.3 Solving the Equation $Gain_s = \tau$

In order to calculate $w_s^*$, the minimum amount of data center bandwidth needed to ensure that the switching condition $Gain_s \geq \tau$ is satisfied, it is essential to solve the equation $Gain_s = \tau$ and reverse the gain formulation (4.2). To this extent, we study the behavior of the gain formulas when $w_s$ varies. Based on this constraint, we identify two exhaustive cases in which the gain equations, as functions of $w_s$, are monotonically decreasing:

- Case A: $(L_s - 1)\, d_{min,s} \geq L_s\, \eta_s\, u_s$
- Case B: $(L_s - 1)\, d_{min,s} \leq L_s\, \eta_s\, u_s$

For each case, we study the behavior of the gain, identify the intervals delimiters and provide, for each interval, the corresponding equation of $w_s^*$ as a function of the switching constraint $\tau$. The complete details about the solution are available in Appendix A. But, tor the sake of clarity, we only include here the solution, presented in (5.1) and (5.2), as follows:

**Case A:** $\boldsymbol{(L_s - 1)\, d_{min,s} \geq L_s\, \eta_s\, u_s}$ When the average download speed of the peers in the swarm $s$ is higher than the upload bandwidth that the whole swarm can provide, the solution $w_s^*$ that satisfies the equation $Gain_s = \tau$, is:

$$w_s^* = \begin{cases} L_s\, d_{min,s}, & \forall \tau \in \left]-\infty, -\frac{\alpha_{bt} d_{min,s}}{F_s}\right] \\ \frac{(1-\tau) F_s\, L_s\, d_{min,s}}{F_s + d_{min,s}\, \alpha_{bt}}, & \forall \tau \in \left[-\frac{\alpha_{bt}\, d_{min,s}}{F_s}, \frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}\,(d_{min,s} - \eta_s\, u_s)}{F_s}\right] \\ \frac{\sqrt{a^2b^2 - 2abc + 4ab + c^2} - ab - c}{2\,b}, & \forall \tau \in \left[\frac{\eta_s u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s u_s)}{F_s}, \frac{L_s - 1}{L_s} - \frac{\alpha_{bt}\eta_s u_s}{(L_s - 1) F_s}\right] \\ \frac{F_s[L_s(1-\tau) - 1]}{\alpha_{bt}}, & \forall \tau \in \left[\frac{L_s - 1}{L_s} - \frac{\alpha_{bt}\, \eta_s\, u_s}{(L_s - 1)\, F_s}, \frac{L_s - 1}{L_s}\right[ \\ \nexists, & \forall \tau \in \left[\frac{L_s - 1}{L_s}, +\infty\right[, \end{cases} \tag{5.1}$$

where: $a = \eta_s\, L_s\, u_s$, $b = \frac{\alpha_{bt}}{F_s\, L_s}$, and $c = \tau$.

**Case B:** $\boldsymbol{(L_s - 1)\, d_{min,s} \leq L_s\, \eta_s\, u_s}$ When the average download speed of the peers in the swarm $s$ is lower than the upload bandwidth the whole swarm can provide, the solution $w_s^*$ that satisfies the equation $Gain_s = \tau$, is:

$$w_s^* = \begin{cases} L_s\, d_{min,s}, & \forall \tau \in \left]-\infty, -\frac{\alpha_{bt}\, d_{min,s}}{F_s}\right] \\ \frac{(1-\tau) F_s L_s d_{min,s}}{F_s + d_{min,s}\, \alpha_{bt}}, & \forall \tau \in \left[-\frac{\alpha_{bt}\, d_{min,s}}{F_s}, \frac{L_s - 1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}\right] \\ \frac{F_s\,[L_s(1-\tau) - 1]}{\alpha_{bt}}, & \forall \tau \in \left[\frac{L_s - 1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}, \frac{L_s - 1}{L_s}\right[ \\ \nexists, & \forall \tau \in \left[\frac{L_s - 1}{L_s}, +\infty\right[. \end{cases} \tag{5.2}$$

## 5.4 Bandwidth Allocation Algorithm

In this section, we present the bandwidth allocation algorithm. The main goal of this algorithm is to optimally manage the data center's limited bandwidth among the seeder nodes. It is also responsible for evaluating for each requested file the most suitable content distribution model: client-server or peer-assisted, based on the current demand load. Each active seeder node in the system is associated with a swarm of clients that are interested in the same file.

It is important to remind here that the default bandwidth distribution protocol is HTTP, and BitTorrent can be also used when the switching conditions previously stated are satisfied. The swarms whose peers are using HTTP as a transfer protocol are referred to as *HTTP swarms* ($S_{HTTP}$ is the set of HTTP swarms) and the ones with peers downloading via BitTorrent are labeled as *BitTorrent swarms* ($S_{BT}$ is the set of BitTorrent swarms).

A swarm will switch to BitTorrent if the following conditions are satisfied:

1. The number of clients in the swarms is higher or equal to 2. In fact, it makes no sense to use BitTorrent with only one client interested in the file.

2. The switch to BitTorrent should satisfy the QoS constraint $\tau$. This means that BitTorrent can be used only when the $Gain_s$ is higher or equal than $\tau$.

3. The amount of data center bandwidth allocated in BitTorrent should be smaller than the one using HTTP. This means that the switch will only take place if the data center would gain in terms of bandwidth.

The algorithm is executed whenever a change affects a swarm $s^* \in S$. This change can be related to a modification in one or more of the parameters of $s^*$, which corresponds to one or more of the following cases:

- A new peer $p^*$ wants to download a file $f_{s^*}$. If the file is already being downloaded by other peers, then $p^*$ will be added to the existing swarm $s^*$. Otherwise, a new swarm $s^*$ will be created containing the single peer $p^*$.

**Algorithm 2** Bandwidth Allocation Algorithm

```
Input $S$          ▷ the set of all the current swarms ($S = S_{HTTP} \cup S_{BT}$)
Input $s^*$        ▷ swarm affected by a change
Input $W$          ▷ the data center's upload bandwidth budget limit
Input $\tau$       ▷ the switching constraint

 1: if $L_{s^*} = 1$ then                                  ▷ $s^*$ is a single-peer swarm
 2:     $w_{s^*} = D_{s^*}$
 3: else if $L_{s^*} > 1$ then                             ▷ $s^*$ has more than one peer
 4:     if $isBT_{s^*} = False$ then                       ▷ $s^*$ is a HTTP swarm
 5:         calculate $w^{bt}_{s^*}$ using (5.1) and (5.2)
 6:         if $w^{bt}_{s^*} \leq D_{s^*}$ then            ▷ switching to BT
 7:             switch from HTTP to BT
 8:             $isBT_{s^*} = True$                        ▷ mark $s^*$ as a BT swarm
 9:             $w_{s^*} = w^{bt}_{s^*}$
10:         else                                           ▷ not switching to BitTorrent
11:             $w_{s^*} = D_{s^*}$
12:         end if
13:     else                                               ▷ $s^*$ is a BT swarm
14:         $w_{s^*} = w^{bt}_{s^*}$ calculated using (5.1) and (5.2)
15:     end if
16: else                                                   ▷ $L_{s^*} = 0$, $s^*$ no longer exists
17:     remove $s^*$ from $S$
18:     if $\sum_{s \in S} w_s + w_{s^*} = W$ then         ▷ the data center was overloaded
19:         for each $s$ in $S_{BT}$ do
20:             $w_s = w_s + \frac{w_s}{\sum_{s \in S_{BT}} w_s} w_{s^*}$   ▷ redistribute $w_{s^*}$ to BT swarms
21:         end for
22:     end if
23: end if

24: if $\sum_{s \in S} w_s > W$ then
25:     for each $s$ in $S$ do
26:         $w_s = \frac{w_s}{\sum_{s \in S} w_s} W$       ▷ scale down all the bandwidth shares
27:     end for
28: end if
```

- A peer $p^*$ leaves a swarm $s^*$. If $p^*$ was not the only peer in the swarm, then the modified swarm will contain a list of the other remaining peers. If $p^*$ was the last peer in $s^*$, then $s^*$ will be removed from $S$.
- The upload or download speed of one or more of the peers in $s^*$ changes.

The algorithm requires the following input parameters: the set of all current swarms $S$, the swarm affected by the change $s^*$, the data center's upload bandwidth limit $W$ and the switching constraint $\tau$. Using these input parameters, the algorithm identifies for each swarm the most suitable download protocol (HTTP or BitTorrent) and calculates the amount of bandwidth to be allocated to the corresponding swarm, as follows:

- If $s^*$ is a single-peer swarm ($L_{s^*} = 1$), then the data center allocates to $s^*$ a share of bandwidth equal to its aggregated download capacity: $w_{s^*} = D_{s^*}$ (*lines 1 and 2*). In this case, the file will be distributed directly from the data center to the single-peer in $s^*$ using HTTP.
- If the number of peers in $s^*$ is strictly higher than 1 (*lines 3 to 12*), then there are two possible cases:
  - If the peers in $s^*$ are using HTTP to download $f_{s^*}$ ($isBT_{s^*}=$ *False*), the algorithm verifies if it is worth switching to BitTorrent. To do so, $w^{bt}_{s^*}$ is calculated according to (5.1) and (5.2). We remind that $w^{bt}_{s^*}$ measures the minimum amount of data center bandwidth required to verify the QoS constraint $\tau$ when using BitTorrent for $s^*$. The algorithm compares later this bandwidth ($w_{s^*}$) with the bandwidth allocated by default to the swarm (which is equal to $D_{s^*}$).
    - If the bandwidth required using BitTorrent is smaller than the one allocated by default ($w^{bt}_{s^*} \leq D_{s^*}$), then the download protocol more suitable for $s^*$ is BitTorrent (*lines 4 to 9*). In this case, a `.torrent` file associated to $f_{s^*}$ is created. All the peers in $s^*$ have to download this `.torrent` and start downloading $f_{s^*}$ via BitTorrent.
    - If the use of BitTorrent requires more bandwidth than HTTP, then it is not worth switching to BitTorrent. In this

case, the data center allocates to $s^*$ a share of bandwidth equal to $D_{s^*}$ (*line 11*).

  - If $s^*$ has already switched to BitTorrent, then the algorithm recalculates $w^{bt}_{s^*}$: the bandwidth needed to maintain the QoS constraint $\tau$ after the changes in $s^*$. The amount of bandwidth allocated to $s^*$ is equal to $w^{bt}_{s^*}$.

- If $s^*$ is an empty swarm ($L_{s^*} = 0$), then the swarm is removed from the swarms' list. If the data center was overloaded before the removal of $s^*$, then the amount of bandwidth that was previously allocated to $s^*$ is redistributed among the BitTorrent swarms (*lines 18 to 22*). This will prevent the data center's bandwidth from being underutilized and will boost the distribution of the files among the BitTorrent swarms.

When the number of simultaneous requests becomes high, the data center might be unable to serve all the swarms at their full speed. In such case, the data center has to scale down all the bandwidth allocations proportionally to the demand (*lines 24 to 28*).

**Complexity of the Algorithm** Our bandwidth allocation algorithm has a complexity of $O(m+n)$, where $n$ is the maximum number of swarms and $m$ is the maximum number of clients. $O(m+n) \approx O(m)$ since $m \geq n$, the number of clients is always higher (or equal) than the total number of swarms. Thus, the algorithm's complexity is linear with the number of clients in the system.

The coordinator keeps in memory the state of the swarms and the corresponding clients during each iteration. This corresponds to a maximum of $n \times k_1 + m \times k_2$ units of storage. $k_1$ (respectively $k_2$) is the size, in units of storage, required to store the essential information about a swarm (respectively a client). $k_1$ and $k_2$ are rather small compared to the size of the files, and they depend on the required information to store for a swarm and a client, respectively. For example, the information required to store for a client are mainly: the client's upload and download bandwidths and the amount of data left to download. For a swarm, it is important for the coordinator to keep a track of: the state of the swarm (the boolean value $isBT_s$) and the list of the clients in the swarm.

To summarize, we can confirm that the algorithm does not require a lot of extra computational and memory resources. This claim is further verified in Section 5.5.4.

## 5.5 Application Scenario: Personal Clouds

In this section, we evaluate the efficiency of the bandwidth allocation algorithm on the U1 trace. We first detail the experiments settings and describe the simulators used for this validation. Later, we present the implications of Algorithm 2 on the U1 trace. After that, we investigate the effects of having bigger shared using two new traces that can be derived from the original U1 trace. Finally, we measure the overhead related to the application of the algorithm in a Personal Clouds scenario.

### 5.5.1 Experimental Settings

To evaluate the efficiency of our proposal, we developed a Python[1] script that simulates the bandwidth allocation algorithm (Algorithm 2) and logs the results in two different log files. The first log file is related to the cloud and it keeps a log of the current state of the cloud. At each timestamp, several parameters are logged including: the amount of needed bandwidth, the amount of bandwidth served, and the current number of swarms and clients. The second log keeps track of the start and end times of each download. The download times are updated at each timestamp according to equations (2.2) and (4.1), for HTTP and BitTorrent swarms, respectively.
In order to evaluate the results, we also developed another script (HTTP-only simulator) that simulates the default behavior of the cloud, in which each download operation is treated individually and the download times are calculated according to equation (2.2). This simulator also keeps similar logs as the algorithm simulator in order to facilitate the comparison of the approaches.

[1] Python Software Foundation: http://www.python.org

### 5.5.2 Implications of the Bandwidth Allocation Algorithm

In this section, we present the results of the application of Algorithm 2 on the U1 trace.

**Measuring the effect of $\tau$** First of all, we run the simulator fixing the upload bandwidth limit at 300 Mbps and varying the switching constraint $\tau$ between $\tau_1 = -1$ and $\tau_8 = 1$, to get a first idea of the performance of the algorithm. We remind here that $\tau_1 = -1$ means that losses in download time up to 100% of the original download time are tolerated. $\tau_8 = 1$ means that only gains in download time equal to at least 100% of he original download time are accepted.

We measure for each simulation, the download time taken by each operation and compare them to the ones measured using the HTTP-only simulator with the same budget limit. It is important to note here that the download times are measured in seconds with a precision of one millisecond. We classify the operations into three different categories: operations that have gained in download time with our algorithm, operations that experienced losses and operations whose download time is left unchanged for both approaches.

Table 5.1 presents the percentages of the operations in each category. We notice that for $\tau \in \{-1, -0.5, -0.2, 0, 0.2, 0.4\}$, more than 82% of the operations benefited from a gain in download time, about 15% kept the same time and only less 3% of them lost in download time. For stricter constrains, $\tau_7 = 0.5$, the percentage of operations with gains drops to 60.36% and the ones with losses grows to reach 14.45%. With $\tau_8 = 1$, the percentage of gains drops to as low as 1.29%.

To make sure that the cumulative gains are higher than the losses, we sum all the download times of all operations for both approaches and calculate the total net gain percentage: $net\_gain\_\%$. Specifically, $net\_gain\_\%$ represents the percentage ratio between the total time gained (or lost) by using the algorithm ($net\_gain\_hours = sum\_http\_hours - sum\_algo\_hours$) and the total download times using HTTP only ($sum\_http\_hours$), as follows:

$$net\_gain\_\% = \frac{net\_gain\_hours}{sum\_http\_hours} \times 100 = \frac{sum\_http\_hours - sum\_algo\_hours}{sum\_http\_hours} \times 100.$$

Table 5.2 presents the total sum of all the download times of all the

**Table 5.1:** Percentages of operations with gains and losses in download time resulted by the algorithm compared to pure HTTP use. The cloud's upload bandwidth budget limit is $W$=300 Mbps.

| | $\tau_1 = -1$ | $\tau_2 = -0.5$ | $\tau_3 = -0.2$ | $\tau_4 = 0$ | $\tau_5 = 0.2$ | $\tau_6 = 0.4$ | $\tau_7 = 0.5$ | $\tau_8 = 1$ |
|---|---|---|---|---|---|---|---|---|
| **% of operations with gain** | 82.87% | 82.87% | 82.89% | 82.9% | 83.43% | 83.53% | 60.36% | 1.29% |
| **% of operations with loss** | 2.23% | 2.23% | 2.23% | 2.31% | 2.48% | 2.85% | 14.45% | 0.24% |
| **% of operations with no difference** | 14.9% | 14.9% | 14.88 % | 14.79% | 14.09% | 13.62% | 25.19% | 98.47% |
| **Total** % | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |

**Table 5.2:** Total sum (in hours) of all the download times for all the operations and the net gain percentage for the algorithm applied on the one-hour sample of the U1 trace. The cloud upload bandwidth budget limit is $W$=300 Mbps.

| | $\tau_1 = -1$ | $\tau_2 = -0.5$ | $\tau_3 = -0.2$ | $\tau_4 = 0$ | $\tau_5 = 0.2$ | $\tau_6 = 0.4$ | $\tau_7 = 0.5$ | $\tau_8 = 1$ |
|---|---|---|---|---|---|---|---|---|
| ***sum_http_hours*** | 2450.2 | 2450.2 | 2450.2 | 2450.2 | 2450.2 | 2450.2 | 2450.2 | 2450.2 |
| ***sum_algo_hours*** | 2001.9 | 2001.67 | 2000.98 | 1997.05 | 1952.73 | 1906.56 | 2064.4 | 2142.57 |
| ***net_gain_hours*** | 448.3 | 448.53 | 449.22 | 453.15 | 497.47 | 543.64 | 385.8 | 307.63 |
| ***net_gain_%*** | **18.3%** | **18.31%** | **18.33%** | **18.49%** | **20.3%** | **22.19%** | **15.75%** | **12.56%** |

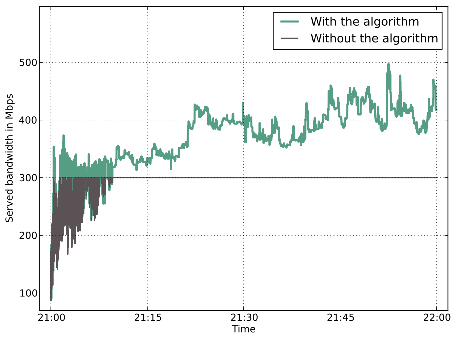
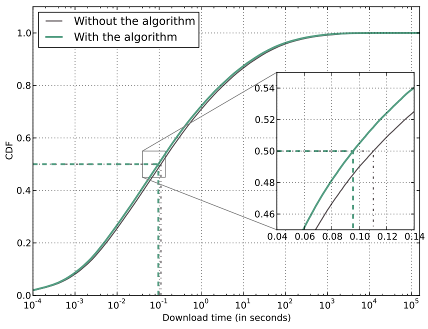


**Figure 5.1:** Amount of bandwidth served to the clients with and without the algorithm. The extra served bandwidth with the algorithm comes from the peers involved in BitTorrent swarms. Settings: $W = 300$ Mbps and $\tau = 0.4$.

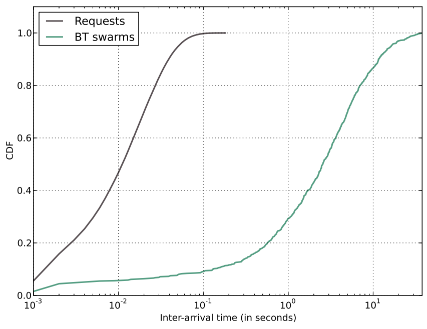
download operations and the net gain percentage based on the U1 one-hour sample. The first row represents the sum of download times using HTTP. It is important to mention here that, for HTTP, this sum depends only on the cloud upload bandwidth budget $W$. Hence, for the fixed bandwidth $W = 300$ Mbps, it is always equal to 2450.2 hours, regardless of the $\tau$ constraint. However, the sum of the download times using the algorithm with a given cloud bandwidth limit depends highly on the switching constraint $\tau$.

In Table 5.2, we compare the results with different $\tau$ values: The first constraint is $\tau_1 = -1$. Under this constraint, we notice that the algorithm performs better than the default HTTP-only strategy, with a net gain in the client's download time equal to 18.3%. With the three following constraints: $\tau_2 = -0.5$, $\tau_3 = -0.2$ and $\tau_4 = 0$, the net gain percentage is also around 18.3%, with slight improvements when $\tau$ gets higher. With a stricter constraint, $\tau_6 = 0.4$, the improvement get higher to reach 22.19%. This improvement is due to the fact that stricter constraints will prevent swarms with negative gains from switching which will result in an increase of the total amount of net gain hours.

However, a very strict constraint ($\tau_7 = 0.5$ and $\tau_8 = 1$) reduces significantly the number of probabilities of switching to BitTorrent, which can affect negatively the performance of the algorithm and limit the benefits of the approach. The measured gain drops in these cases to 15.75% and



**(a)** Download time



**(b)** Inter-arrival time

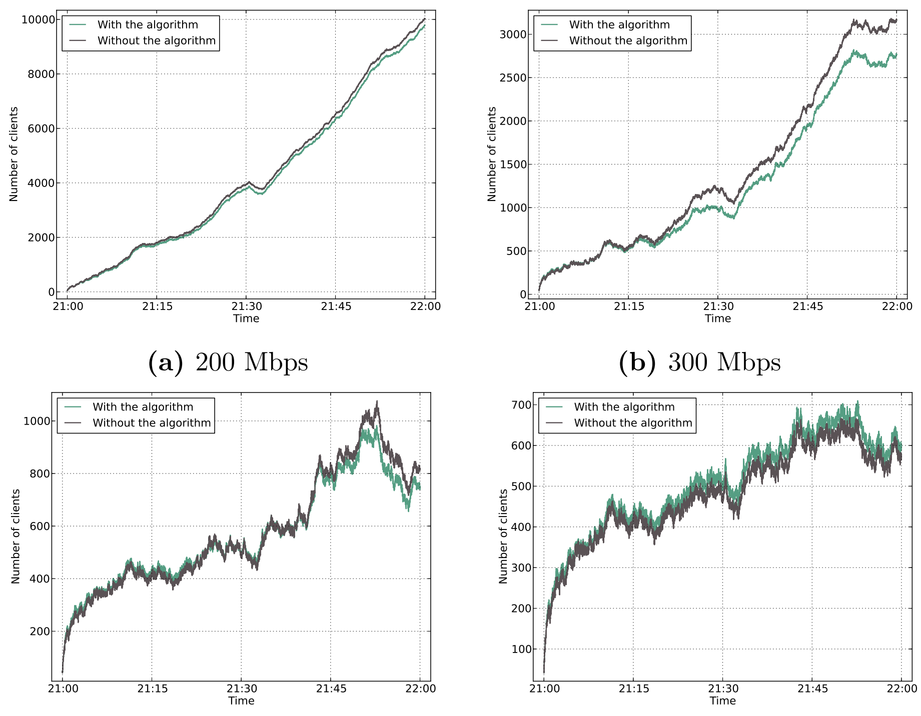
**Figure 5.2:** CDF download times and inter-arrival time of requests and BT swarms with and without the algorithm. Settings: $W = 300$ Mbps and $\tau = 0.4$.

12.56%, respectively.

**Measuring the peer's contribution** To measure the efficiency of the algorithm for a specific configuration, we fix the switching constraint to $\tau = 0.4$ and we suppose that the cloud's upload budget limit is $W = 300$ Mbps.

Figure 5.1 shows the efficiency of taking advantage of the upload speed of the peers. It compares the amount of bandwidth served by the cloud to the clients without using the algorithm (all files are distributed via HTTP) versus the total amount that becomes available when using the algorithm. This latter includes, in addition to the cloud's upload limit, the upload speed of the clients who switched to BitTorrent. We notice that the peers' contribution can reach up to 60% of the total cloud's budget.

**Download and inter-arrival times** With the same configuration as before ($W = 300$ Mbps and $\tau = 0.4$), we can see in Table 5.1 that the total number of operations switched to BitTorrent represents only 2.45% of the operations with multiple downloads. This corresponds to less than 1% of the total number of operations. Despite this limited number of switched operations, we notice in Figure 5.2a that the download times are reduced using the algorithm. As a matter of fact, the average download time without using the algorithm is 39.11 seconds. This time is reduced



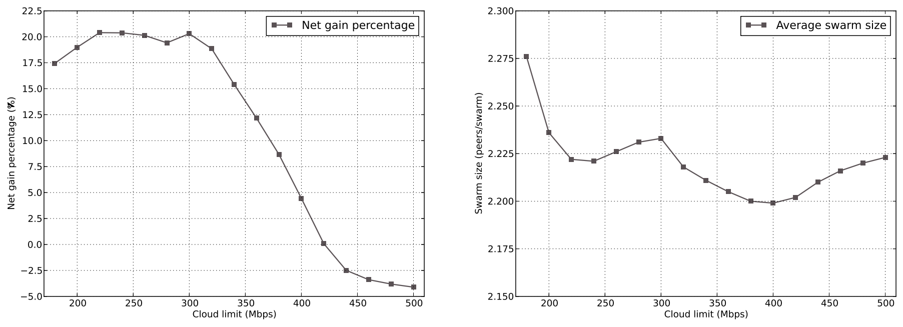
**(a)** 200 Mbps **(b)** 300 Mbps

**(c)** 400 Mbps **(d)** 500 Mbps

**Figure 5.3:** Number of simultaneous clients present in the system with and without the algorithm. Settings: $w = 200, 300, 400$, and $500$ Mbps; $\tau = 0.4$

by 22.19% using the algorithm to only 30.43 seconds. Figure 5.2b presents the CDF of the inter-arrival times of requests and BitTorrent swarms. The average inter-arrival rate of the download requests (time between each arrival of a download request into the system and the next) is 0.0159 seconds. The average inter-arrival rate of the BitTorrent swarms (time between each creation of a BitTorrent swarm and the next) is 4.5702 seconds.

**Simultaneous clients** After evaluating the general performance of the algorithm, we study the effect of the bandwidth limit $W$ on the total number of simultaneous clients. Figure 5.3 compares the number of simultaneous



**(a)** Net gain percentage **(b)** Average BitTorrent swarms' size

**Figure 5.4:** Net gain percentage and average size of the BitTorrent swarms for different cloud bandwidth limits ranging between 180 and 500 Mbps. The switching constraint considered here is $\tau = 0.2$.

clients in the HTTP-only mode and using the algorithm for four different values of $W$: 200, 300, 400 and 500 Mbps. It is important to mention here that in our simulator, clients do not stay as seeders in the system. They leave as soon as they finish downloading the requested files. When comparing the number of simultaneous peers using each of the approaches, we note that a lower number of simultaneous peers means that the clients are downloading faster which proves that the corresponding approach is more efficient. We notice that with limited bandwidth budget (200 and 300 Mbps), the algorithm performs better than pure HTTP. This due to the fact that when the cloud has a very limited bandwidth budget, the share allocated to each swarm will be small. Therefore, the HTTP download time will be "high" and the overhead of switching to BitTorrent will be negligible. However, the higher the cloud bandwidth gets, the bigger the overhead becomes compared to the download time in HTTP. This explains the degrade in the algorithm's performance when the cloud's bandwidth budget becomes quite high (400 Mbps).

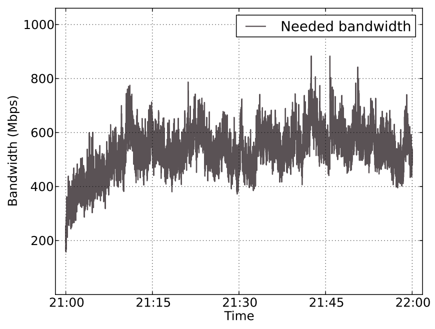
**Measuring the effect of $W$** Figure 5.4 summarizes, for different values of $W$ ranging from 180 Mbps to 500 Mbps, the net gain percentage and the average size of BitTorrent swarms. Figure 5.4a plots the evolution of the net gain percentage with the cloud's bandwidth. Similar to the aforementioned conclusions, when the bandwidth is small (lower than 320 Mbps), the net gain percentage in download time of the clients is important (between 17.5% and 21%). However it gets lower with the increasing budget of the cloud, until reaching negative values when the cloud's bandwidth is higher than 420 Mbps. This confirms our previous conclusions that the algorithm is more efficient when the cloud has very limited bandwidth resources. Figure 5.4b presents the average number of peers in BitTorrent swarms. We notice that most of the BitTorrent swarms are very small with an average size of about 2.22 peers per swarm. This can be due to the limited sharing in U1 system and to the fact that most of U1 users are using the service for data backup only.

### 5.5.3 Modified Trace: Bigger Shared Files

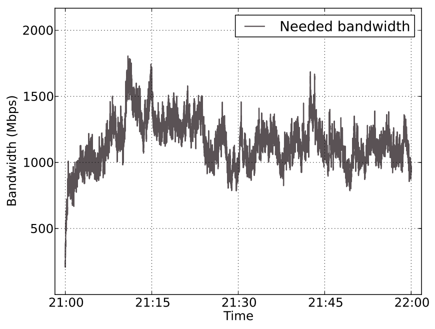
Even though the U1 trace has limited sharing and very small files (most of the files are smaller than 1 MB), we could achieve relatively important improvements in the system's performance. To further validate our proposal, we modify the trace in order to have bigger shared files. Our idea is to keep the same arrival pattern of the peers and just increase the size of the files that were downloaded more than once. We obtain two different modified traces:

- *trace_1*: This trace preserves the same arrival pattern as in the original trace, but we increase the size of the files smaller than 1 MB by 1 MB. For instance, if a file $f_s$ is downloaded more than once in the original trace sample and has a file size of 100 KB, then, in *trace_1*, the same file would be 1 MB (1024 KB) bigger, that is 1124 KB. We chose this value (1 MB) because it represents the mean file size of all the files in the original trace.

- *trace_2*: This trace is obtained the same way as *trace_1*. We choose a bigger limit on size equal to 5 MB. This means that *trace_2* also preserves the same arrival pattern as in the original trace, but here we increase the size of the files smaller than 5 MB by 5 MB. For instance, if a file $f_s$ is downloaded more than once in the original trace sample and has a file size of 1 MB, in *trace_2*, the same file would be 5 MB bigger, that is 6 MB.



**(a)** *trace_1*



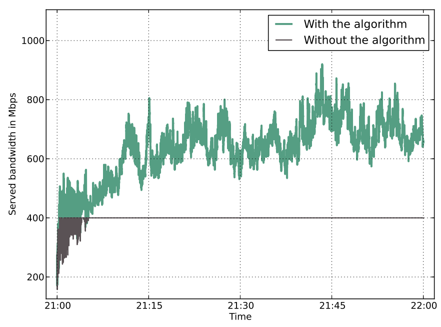
**(b)** *trace_2*

**Figure 5.5:** New bandwidth requirements of the modified traces

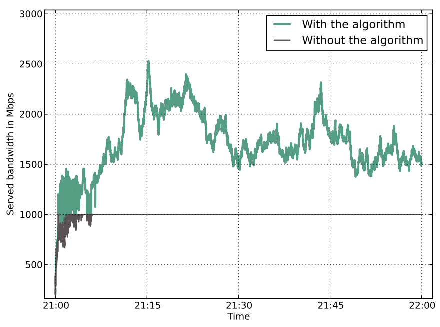
**Table 5.3:** Comparison of the results with the three different traces. The experiments settings are the followings: $\tau = 0.4$ for all the traces, $W$ is 300 Mbps for the original trace, 400 Mbps for *trace_1*, and 1000 Mbps for *trace_2*.

| | **original trace** | ***trace_1*** | ***trace_2*** |
|---|---|---|---|
| **Number of operations affected by the change** | 0 | 69,286 | 65,992 |
| **(% of the total number of operations)** | (0%) | (30.72%) | (29.26%) |
| **Number of operations switched to BitTorrent** | 1,734 | 36,727 | 43,997 |
| **(% of the total number of operations)** | (0.76%) | (16.28%) | (19.5%) |
| **Number of BitTorrent swarms created** | 786 | 9,324 | 10,763 |
| **Average BitTorrent swarm size (peer/swarm)** | 2.21 | 3.93 | 4.08 |
| **Average inter-arrival time of BitTorrent swarms (s)** | 4.570 | 0.386 | 0.334 |
| **Average download time without the algorithm (s)** | 39.11 | 106.09 | 186.82 |
| **Average download time with the algorithm (s)** | 30.43 | 52.77 | 61.08 |
| ***net_gain_%*** | **22.19%** | **50.26%** | **67.30%** |

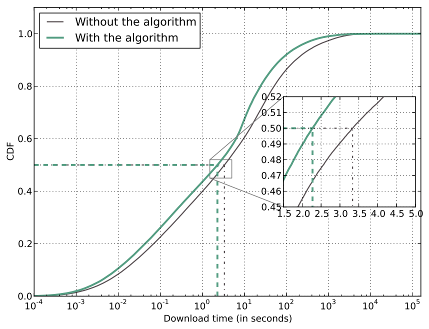
**Bandwidth requirements** Clearly, when we increase the size of some files, the amount of bandwidth needed to distribute the requested files to the peers will increase too. Figure 5.5 presents the new required bandwidth for both traces and shows that *trace_1* and *trace_2* require clearly more bandwidth compared to the original requirements (Figure 2.8). We apply later our algorithm on both traces and compare the results. We use the same switching constraint $\tau = 0.4$ and we fix the cloud's upload budget limit $W$ to 400 Mbps and 1000 Mbps for *trace_1* and *trace_2* respectively.



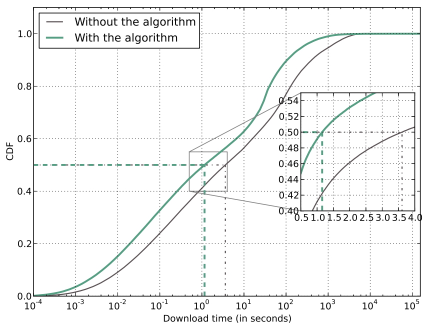
**(a)** *trace_1*: Served bandwidth



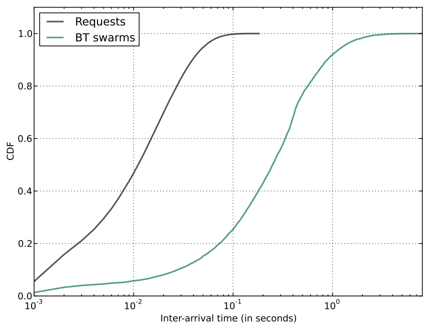
**(b)** *trace_2*: Served bandwidth



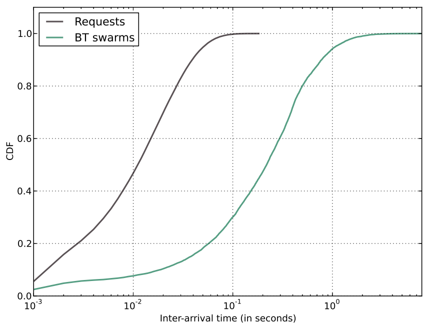
**(c)** *trace_1*: CDF download times



**(d)** *trace_2*: CDF download times



**(e)** *trace_1*: CDF inter-arrival times



**(f)** *trace_2*: CDF inter-arrival times

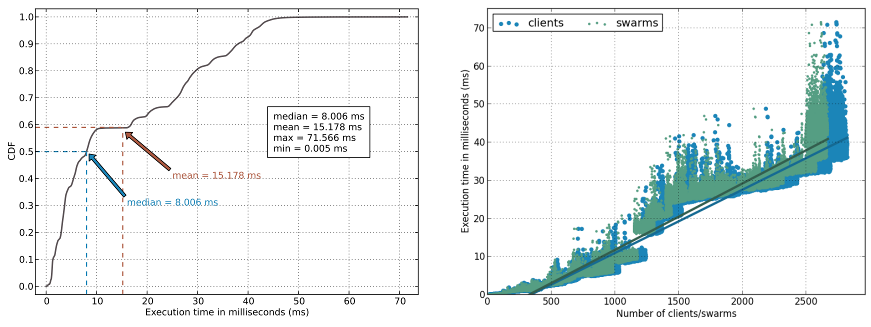
**Figure 5.6:** Results using the modified traces

**Results** Figure 5.6 and Table 5.3 summarize the results with and without the algorithm. As we can see in Figures 5.6a and 5.6b, the amount of bandwidth contributed by the BitTorrent clients can reach up to 100% of the cloud's initial limit. In addition, we notice important improvements in the net gain percentage that increases from about 22% for the original trace to reach over 50% when the files are bigger than 1 MB and more than 65% when the files are bigger than 5 MB. In fact, increasing the file sizes results in increased probability of switching to BitTorrent. Actually, the percentage of operations switched to BitTorrent grows from 0.76% in the original trace and reaches 19.5% in *trace_2*. This leads to a noticeable decrease in the inter-arrival time of BitTorrent swarms as seen in Figures 5.6e and 5.6f. The frequency of creation of new BitTorrent swarms increases from 0.21 swarm per second for the original trace to 2.59 swarms per second for *trace_1* and reaches 2.98 swarms per second for *trace_2*. Similarly, the average size of the swarms increases from 2.21 peers per swarm to reach about 4.08 peers per swarm when the sizes of the shared files become bigger.

### 5.5.4 Performance of the Bandwidth Allocation Algorithm

To measure the efficiency of the algorithm, we measure the time needed to simulate the original trace. The simulation of the whole trace (more than $225,000$ operations) took around 82 minutes until all the downloads have finished. We also measure the time needed to calculate the bandwidth distribution for each timestamp (with the arrival of each new download operation). This time corresponds to one execution round of the algorithm and we refer to it as the execution time. Figure 5.7a presents the CDF of this execution time. It varies between 0.005 and 71.566 milliseconds, depending on the number of clients present in the system. The mean and median execution times are 15.178 and 8.006 milliseconds, respectively.

To measure the effect of the number of clients and swarms on the execution time at each round of the algorithm, we depict in Figure 5.7b the scatter plot of the execution times as a function of the number of clients/swarms, along with the corresponding polynomial regression of degree $d = 1$. The slopes of these lines are equal to 0.016 and 0.017 for the number of clients and the number of swarms respectively.



**(a)** CDF execution time

**(b)** Execution time per number of swarms/clients

**Figure 5.7:** Algorithm's perfomance during the trace simulation

**Summary** Based on the U1 trace, we can confirm that even though the trace involves limited sharing, the bandwidth allocation algorithm achieves important improvements in the measured download times that can exceed 22%. This improvement depends mainly on the cloud's bandwidth budget $W$ and the fixed QoS constraint $\tau$. We notice that the algorithm performs better when the cloud's resources are limited and when the QoS constraint is chosen reasonably.

To validate further our proposal, we modified the original trace in order to have bigger shared files. The performance of the algorithm improve significantly with the increase in file size to reach over 65% of the download time experienced by the peers.

Finally, we measure the performance of the algorithm in terms of execution time on a regular desktop machine. We prove that the overhead needed for each iteration is relatively small. The overall performance would also improve if the algorithm is implemented on a more powerful machine.

## 5.6 Conclusions

In this chapter, we propose a bandwidth allocation algorithm that can be implemented in Personal Cloud systems with limited bandwidth budget. Based on the demand on the cloud and the load on each file, the cloud server is able to decide whether to use a client-server approach or a peer-assisted one to distribute that file. Our proposed algorithm for the management of cloud bandwidth achieves important improvements in terms of download time for the clients, even though in our simulator's implementation we were "stricter" on BitTorrent than HTTP. In fact, we used an estimation of the download time in HTTP that does not take into account the protocol's overheads. However, on the other hand, we added to BitTorrent the potential latency of the peers discovery phase and the delay that can be caused by pieces unavailability. Moreover, we considered the "worst case scenario" where the peers leave the system as soon as they finish the file download, while in reality, the synchronization process works always in the background without the user being aware of it. This means that it is more probable that the peers will stay longer, even after finishing the download and contribute more to the system. Despite that, the results prove that the use of BitTorrent in Personal Clouds can help the clients gain in download time, especially when the bandwidth resources of the seed are limited. In such conditions, the net gain percentage in the download time of all the peers exceeds 20% of their download time in most cases, based on a real trace of the U1 system.

The original U1 trace has limited sharing and very small files. For this reason, we modified it in order to have bigger shared files. The application of the algorithm on the modified traces results in important improvements in the download time that exceed 65% of the original download time of all the peers.

Nevertheless, several extensions can be added to the algorithm. For instance, it is possible to consider two different values of the switching constraint based on the load of the seed: $\tau_{overloaded}$ and $\tau_{not\_overloaded}$. This way, strict constraints can be put when the seed is not overloaded and loosened them up when the load on the seed increases.

# 6
# Cross-swarm Bundling

In this chapter, we propose cross-swarm bundling to mitigate the problem of lack of simultaneous download requests on the same file which limits the benefits of using BitTorrent. Cross-swarm bundling is the process of merging two swarms together into a single one. We prove in this chapter that this technique can have a positive effect on the Quality of Experience (QoE) in Personal Clouds in particular, and data centers in general. We start the chapter by presenting the concept of cross-swarm bundling and describe the considered bundling model. Next, we prove through experimentation and an average case analysis that bundling can effectively improve the QoE. After that, we present a methodology to implement bundling in data centers, based on graph matching techniques. Finally this methodology is evaluated in Personal Clouds using the U1 trace. The results show that cross-swarm bundling is useful to improve the QoE when the cloud's available outgoing bandwidth is limited.

*The results presented in this chapter are published in [19]*

## 6.1 Introduction

The previous chapters have proven BitTorrent to be effective in decreasing the outbound traffic, despite the small size of data flows and swarms. By benefiting from the common interest of peers in the same file, we provided a methodology to exploit their aggregate upload bandwidth to offload the data center from doing all the serving. However, the results of this approach remain limited, as the number of clients simultaneously interested in the same file is generally small, which poses a limit on the maximum benefits of BitTorrent. This chapter is dedicated to mitigate this problem, by answering the third question raised in the first chapter: *How to increase the contribution of the clients?*

A good solution to this problem is to "inflate" the swarms of clients by grouping a set of diverse contents into a single `.torrent` file. This is known as *bundling* in BitTorrent parlance, and was originally proposed to mitigate the problem of availability in unpopular torrents [87, 63]. In data centers, low content availability is not an issue, as the central servers are always available to serve the files. However, we prove in this chapter that *cross-swarm bundling* can have a positive effect on the Quality of Experience (QoE) in Personal Clouds in particular, and data centers in general. The QoE, also known as the perceived quality of service, measures the user's satisfaction with the provided service. Several parameters can interfere in evaluating the QoE. In this chapter, we focus on the download time observed by the clients. For each request, we compare the download times measured using HTTP only and using bundling. The approach that entails less download time is the one that offers a better QoE.

The goal of this chapter is to maximize the QoE for users, which requires an answer to the questions of *how to decide which swarms should be grouped together among the large set of swarms, and which are the criteria for making such a decision?* As far as we know, we are the first to provide evidence that bundling can be useful to diminish the download time. More specific to our problem is the investigation of the benefits of inflating swarms with HTTP users or even of the merge of two HTTP users to create a BitTorrent swarm.

The remainder of the chapter is organized as follows: First, we start by explaining the concept of bundling and presenting the cross-swarm bundling model in Section 6.2. Second, we measure in Section 6.3, through experimentation, the effects of bundling on the download time for the clients, and prove that this approach can improve the QoE. Section 6.4 presents an average case analysis that predicts the efficiency of bundling in a data center, based on the characteristics of its clients. Next, a methodology to implement bundling in data centers is presented in Section 6.5. This methodology is based on graph matching techniques, and it is later evaluated in Section 6.6 on Personal Clouds using the U1 trace. The results show that cross-swarm bundling is useful to improve the QoE, when the data center's available outgoing bandwidth is limited. Finally, Section 6.7 concludes the chapter.

## 6.2 Cross-Swarm Bundling Model

*Cross-swarm bundling* is the process of merging two[1] swarms, say $s_i$ and $s_j$, together into a single swarm $s_i \cup s_j$ (see Figure 6.1). The files $f_{s_i}$ and $f_{s_j}$, corresponding to $s_i$ and $s_j$, respectively, are grouped together and sent via BitTorrent to the resulting swarm which contains all the peers from both $s_i$ and $s_j$. Once a client of a bundled swarm finishes downloading both files, he can leave the swarm. The amount of data center bandwidth allocated to the bundle is equal to the sum of the amounts allocated to each swarm individually before bundling. For instance, if the data center was allocating $w_{s_i}$ to $s_i$ and $w_{s_j}$ to $s_j$ before bundling, then the bandwidth allocated to the bundled swarm will be equal to $w_{s_i} + w_{s_j}$.

To evaluate the efficiency of bundling on $s_i$ and $s_j$, we compare the download times before and after bundling. We denote by $t_{s_i}^{before}$ and $t_{s_j}^{before}$ the expected end download times for $s_i$ and $s_j$, respectively, before the application of bundling.

$t_{s_i}^{before}$, respectively $t_{s_j}^{before}$, depends on the protocol being used by $s_i$, respectively $s_j$. For example, if the peers in $s_i$ are using HTTP, then $t_{s_i}^{before} = t_{s_i}^{http}$ and it is calculated using (2.2). Otherwise, when the download protocol is BitTorrent, $t_{s_i}^{before} = t_{s_i}^{bt}$, as in (4.1). The same applies for $s_j$.

To this extent, we distinguish between three bundling variants, depending on the download protocol adopted by the grouped swarms.

[1]For the sake of simplicity, we only consider bundling two swarms at a time.

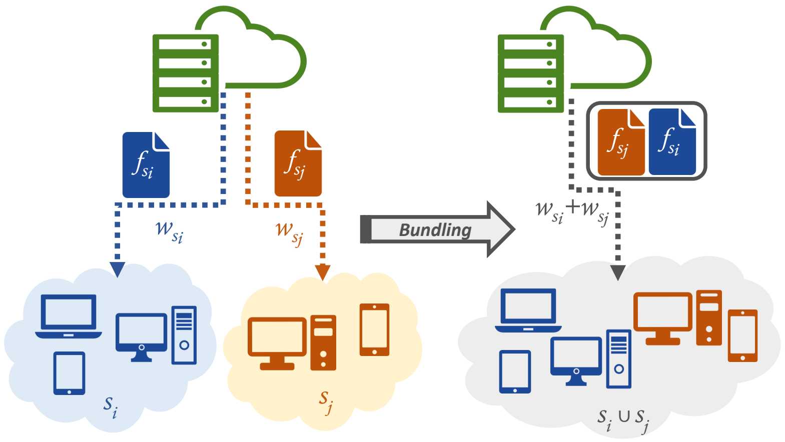


**Figure 6.1:** Cross-swarm bundling scenario

- **HTTP-swarm bundling**: an HTTP swarm is grouped with another HTTP swarm. In this case, bundling implies switching from HTTP to BitTorrent, and thus, will increase the number of BitTorrent swarms at the expense of HTTP ones.
- **BitTorrent-swarm bundling**: a BitTorrent swarm is bundled with another BitTorrent swarm. This is the most known variant of bundling and it has been widely studied in the literature [63, 79, 87, 129]. Bundling BitTorrent swarms will result in bigger swarms, resulting in more collaboration between the peers.
- **Hybrid-swarm bundling**: an HTTP swarm is bundled with a BitTorrent swarm. Hybrid-swarm bundling implies having more, bigger BitTorrent swarms and fewer HTTP ones.

To the best of our knowledge, only BitTorrent-swarm bundling was studied in the literature. We are the first to propose and evaluate the two other variants: HTTP-swarm and hybrid-swarm bundling.

$t^{after}_{s_i,s_j}$ is the expected end download time after the swarms are bundled. It is estimated using (4.1), since the download protocol used for the bundled swarms is always BitTorrent. The file size considered to calculate $t^{after}_{s_i,s_j}$ is $F_{s_i} + F_{s_j}$. The data center's outgoing bandwidth allocated to the bundle is $w_{s_i} + w_{s_j}$, that is the aggregation of the bandwidth that were allocated to each swarm separately.

## 6.3 Bundling Can Reduce Download Times

The idea of bundling two different swarms in order to improve the QoE might seem controversial at first sight. To quantify the real effects of this approach, we ran experiments comparing both approaches (separate versus bundled swarms) in a campus scenario, and compare the measured download times of different peers for each approach.

**Experimental settings** We prepared the set of experiments listed in Table 6.1, using all the combinations of swarms of sizes between 1 and 3 peers downloading small files: 1, 2 and 3 MB. The sizes of the swarms and the files were intentionally chosen to be small, in order to represent the typical swarms in a Personal Cloud [20, 21, 60]. We measured the download times for each of these swarms using HTTP and BitTorrent and then, for each combination of two swarms, we measured the download times when they are bundled together.

For HTTP transfers, we used HFS[2]: HTTP File Server, which is a file sharing server that supports bandwidth control. BitTorrent transfers were performed using the Vuze[3] Java BitTorrent client which is available under the GNU General Public License. The clients considered in these experiments were homogeneous and had each an upload speed of 512 Kpbs and a download speed equal to 1 Mbps. The original content provider (the server in case of HTTP transfer and the original seed in the case of BitTorrent) had an upload speed limit $w_s = 1$ Mbps, reserved for each file transfer. When the files were bundled and the swarms were merged together, this bandwidth limit was of 2 Mbps, the sum of bandwidths that would be allocated to each swarm separately.

**Effects of bundling on the download time** Figure 6.2 represents the experimental results with all the download times measured for each experiment. We notice that generally, the download time with bundling is

[2]HFS: HTTP File Server `http://www.rejetto.com/hfs/`
[3]Vuze: `http://www.vuze.com/`

**Table 6.1:** Bundling experiments list.

| **Exp id** | **Settings** | | | | | | **Exp id** | **Settings** | | | | | | **Exp id** | **Settings** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $s_1$ | | $s_2$ | | $s_1 \cup s_2$ | | | $s_1$ | | $s_2$ | | $s_1 \cup s_2$ | | | $s_1$ | | $s_2$ | | $s_1 \cup s_2$ | |
| | $L_{s_1}$ | $F_{s_1}$ | $L_{s_2}$ | $F_{s_2}$ | $L_{s_{12}}$ | $F_{s_{12}}$ | | $L_{s_1}$ | $F_{s_1}$ | $L_{s_2}$ | $F_{s_2}$ | $L_{s_{12}}$ | $F_{s_{12}}$ | | $L_{s_1}$ | $F_{s_1}$ | $L_{s_2}$ | $F_{s_2}$ | $L_{s_{12}}$ | $F_{s_{12}}$ |
| **1** | 1 | 1 | 1 | 1 | 2 | 2 | **16** | 2 | 1 | 2 | 3 | 4 | 4 | **31** | 2 | 2 | 2 | 2 | 4 | 4 |
| **2** | 1 | 1 | 2 | 1 | 3 | 2 | **17** | 2 | 1 | 3 | 3 | 5 | 4 | **32** | 2 | 2 | 3 | 2 | 5 | 4 |
| **3** | 1 | 1 | 3 | 1 | 4 | 2 | **18** | 3 | 1 | 3 | 1 | 6 | 2 | **33** | 2 | 2 | 1 | 3 | 3 | 5 |
| **4** | 1 | 1 | 1 | 2 | 2 | 3 | **19** | 3 | 1 | 1 | 2 | 4 | 3 | **34** | 2 | 2 | 2 | 3 | 4 | 5 |
| **5** | 1 | 1 | 2 | 2 | 3 | 3 | **20** | 3 | 1 | 2 | 2 | 5 | 3 | **35** | 2 | 2 | 3 | 3 | 5 | 5 |
| **6** | 1 | 1 | 3 | 2 | 4 | 3 | **21** | 3 | 1 | 3 | 2 | 6 | 3 | **36** | 3 | 2 | 3 | 2 | 6 | 4 |
| **7** | 1 | 1 | 1 | 3 | 2 | 4 | **22** | 3 | 1 | 1 | 3 | 4 | 4 | **37** | 3 | 2 | 1 | 3 | 4 | 5 |
| **8** | 1 | 1 | 2 | 3 | 3 | 4 | **23** | 3 | 1 | 2 | 3 | 5 | 4 | **38** | 3 | 2 | 2 | 3 | 5 | 5 |
| **9** | 1 | 1 | 3 | 3 | 4 | 4 | **24** | 3 | 1 | 3 | 3 | 6 | 4 | **39** | 3 | 2 | 3 | 3 | 6 | 5 |
| **10** | 2 | 1 | 2 | 1 | 4 | 2 | **25** | 1 | 2 | 1 | 2 | 2 | 4 | **40** | 1 | 3 | 1 | 3 | 2 | 6 |
| **11** | 2 | 1 | 3 | 1 | 5 | 2 | **26** | 1 | 2 | 2 | 2 | 3 | 4 | **41** | 1 | 3 | 2 | 3 | 3 | 6 |
| **12** | 2 | 1 | 1 | 2 | 3 | 3 | **27** | 1 | 2 | 3 | 2 | 4 | 4 | **42** | 1 | 3 | 3 | 3 | 4 | 6 |
| **13** | 2 | 1 | 2 | 2 | 4 | 3 | **28** | 1 | 2 | 1 | 3 | 2 | 5 | **43** | 2 | 3 | 2 | 3 | 4 | 6 |
| **14** | 2 | 1 | 3 | 2 | 5 | 3 | **29** | 1 | 2 | 2 | 3 | 3 | 5 | **44** | 2 | 3 | 3 | 3 | 5 | 6 |
| **15** | 2 | 1 | 1 | 3 | 3 | 4 | **30** | 1 | 2 | 3 | 3 | 4 | 5 | **45** | 3 | 3 | 3 | 3 | 6 | 6 |

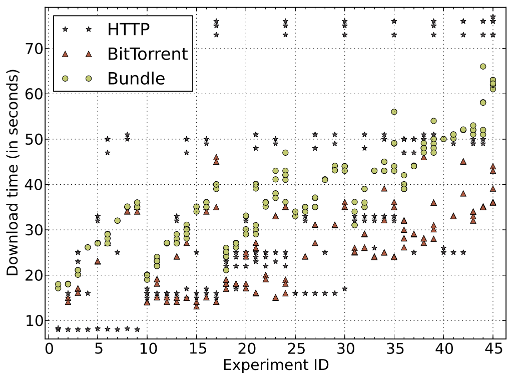
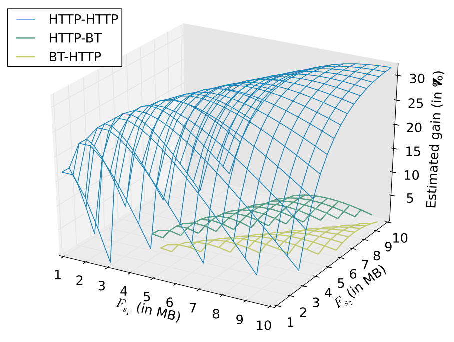


**Figure 6.2:** Separate versus aggregated swarms: Measured download times for small swarms ($L_{s_i} \in \{1, 2, 3\}$) downloading small files ($F_{s_i} \in \{1, 2, 3 \text{ MB}\}$).

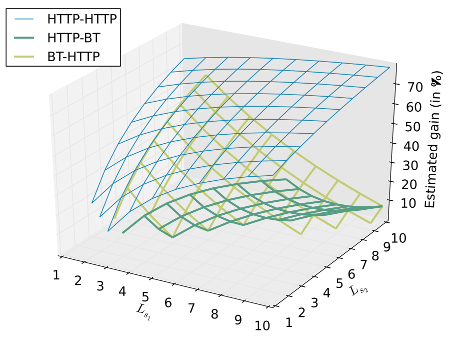
higher than the regular downloads via HTTP or BitTorrent. Nevertheless, there some cases when the measured download times with bundling are lower than with separate swarms, especially when both swarms were using HTTP before bundling (Experiments 36 and 45, for example). For the swarms which were already using BitTorrent, we notice that, in most cases, bundling does not improve the download time.

More concretely, we take the example of two identical swarms $s_1$ and $s_2$, each composed of 3 peers downloading a 3 MB file (Experiment 45 in Figure 6.2 and Table 6.1). When these swarms are downloading the files separately using HTTP (respectively BitTorrent), the experienced download times for the peers in $s_1$ and $s_2$ range from 73 to 77 seconds (respectively 36 to 44 seconds). When $s_1$ and $s_2$ are bundled into a swarm composed of the same 6 peers downloading a 6 MB file, the experienced download times are between 61 and 63 seconds. This means that in the case of two HTTP swarms, bundling has resulted in an improvement in download time higher than 10% of the original time, for all the peers. However, when the swarms were using BitTorrent, bundling has caused a loss in download time of around 35%.

**Effects of bundling on the redundant data sent** In addition to the improvement in download time with HTTP swarms, we notice that the



(a) $L_{s_1} = L_{s_2} = 2$ peers



(b) $F_{s_1} = F_{s_2} = 1$ MB

**Figure 6.3:** Estimated gain in download times for different combinations of swarms.

use of bundling has reduced the amount of redundant data sent from the original content provider. In fact, for the same experiment (Experiment 45), and with 2 separate HTTP swarms, the content provider has to send the size of the file $f_{s_1}$ (respectively $f_{s_2}$) to each of the 3 peers in both $s_1$ (respectively $s_2$). This means that the total amount of data sent by the content provider (without considering package losses) is equal to $3 \times 3 + 3 \times 3 = 18$ MB.
When the swarms are bundled, the amount of data to be sent by the content provider is doubled to 36 MB, which corresponds to sending a 6 MB file to 6 peers. Nevertheless, after bundling, the measured peers' contribution to the aggregated swarm is equal to 21.33 MB, which represents nearly 60% of the total data to be sent (60% of the 36 MB). The content's provider contribution in this case is estimated to $36 - 21.33 = 14.67$ MB. Thus, compared to separate HTTP transfers, we can confirm that bundling can reduce the load on the original content provider, and in this case it has eliminated over 18% of the data transfer cost on the provider's side.

**Evaluating bundling variants** Figure 6.3 presents an overview of the combinations of swarms to bundle that can lead to an improvement in download time. Each swarm $s_i$ is defined by the number of peers $L_{s_i}$ and the size $F_{s_i}$ of the corresponding file $f_{s_i}$. In each of the sub-figures, we fix one of these two parameters and evaluate the estimated gain values for each combination, taking into account all the possible download protocol combinations. We notice that the combinations where both swarms use HTTP before bundling lead to the highest improvements in download times. Nevertheless, we notice that even some HTTP-BitTorrent combinations can entail positive values of the gain when the file sizes (Figure 6.3a) and the swarm sizes (Figure 6.3b) get bigger. However, we notice a total absence of positive gains for BitTorrent-swarm bundling combinations.

**Summary** Based on real experiments, we can confirm that cross-swarm bundling can reduce download time even in the adverse scenario where files are small. But not only this, it also helps in reducing the outgoing bandwidth on the content provider's side thanks to the contribution of peers. A key observation is that *bundling two HTTP swarms performs better than any other type of bundling* and that *the gain in download time tends to increase when the sizes of the swarms and/or the files increase.*

## 6.4 Average Case Analysis

From a data center's perspective, it is important to evaluate the efficiency of bundling before implementing a bundling mechanism. In this section, we focus on the case of bundling two HTTP swarms, the most common one in Personal Clouds. We propose the following notation related to the average characteristics of a swarm in the system:

- $\bar{L}$: average size of a swarm,
- $\bar{w}$: average data center upload bandwidth allocated to a swarm,
- $\bar{F}$: average file size,
- $\bar{u}$: average upload speed of the clients in a swarm,
- $\bar{\eta}$: average value of the efficiency of file sharing,
- $d_{min}$: minimum download speed of the clients,
- $\bar{t}^{before}$: average download time before bundling,
- $\bar{t}^{after}$: average download time after bundling.

Following this notation, we can estimate $\bar{t}^{before}$ and $\bar{t}^{after}$, using (2.2) and (4.1), as follows:

$$\bar{t}^{before} = \frac{\bar{F}}{\min\left\{d_{min}, \frac{\bar{w}}{\bar{L}}\right\}}. \tag{6.1}$$

$$\bar{t}^{after} = \frac{2\bar{F}}{\min\left\{d_{min}, \frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}, 2\bar{w}\right\}}. \tag{6.2}$$

To compare both approaches, the service provider has to identify the bottleneck limiting the transfer of the files from the storage nodes to the end users. Using (6.1) and (6.2), we study all the possible values of $\min\left\{d_{min}, \frac{\bar{w}}{\bar{L}}\right\}$ and $\min\left\{d_{min}, \frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}, 2\bar{w}\right\}$ and we compare $\bar{t}^{before}$ and $\bar{t}^{after}$ for each possible case of the bottleneck, as follows:

1. **<u>Case 1:</u>** $d_{min} \leq \frac{\bar{w}}{\bar{L}}$ and $d_{min} \leq \min\left\{\frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}, 2\bar{w}\right\}$. When the transfer bottleneck is the download speed of the peers, bundling is not efficient.
2. **<u>Case 2:</u>** $\frac{\bar{w}}{\bar{L}} \leq d_{min}$ and $2\bar{w} \leq \min\left\{d_{min}, \frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}\right\}$. When the data center's bandwidth is the bottleneck, bundling is efficient, if and only if the average swarm is composed of at least 2 peers.
3. **<u>Case 3:</u>** $\frac{\bar{w}}{\bar{L}} \leq d_{min}$ and $\frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}} \leq \min\{d_{min}, 2\bar{w}\}$. When the upload capacity of the swarm is the bottleneck, bundling is efficient, if and only if the swarm's upload capacity is higher than the amount of data center bandwidth allocated to the swarm.
4. **<u>Case 4:</u>** $\frac{\bar{w}}{\bar{L}} \leq d_{min}$ and $d_{min} \leq \min\left\{\frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}, 2\bar{w}\right\}$. When the data center's bandwidth and the download speed of the peers are the bottlenecks before and after bundling respectively, bundling is efficient, if and only if the download capacity of the clients is higher than the data center's bandwidth.

*Proof.* The proof is organized into the same cases as above. We consider bundling efficient, if and only if, the download time resulted from bundling is lower than the original download time. This means that bundling is efficient, if and only if, $\bar{t}^{before} > \bar{t}^{after}$.

1. <u>Case 1:</u> $d_{min} \leq \frac{\bar{w}}{\bar{L}}$ and $d_{min} \leq \min\left\{\frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}, 2\bar{w}\right\}$.

$$\begin{cases} d_{min} \leq \frac{\bar{w}}{\bar{L}} & \Longrightarrow \bar{t}^{before} = \frac{\bar{F}}{d_{min}} \\ d_{min} \leq \min\left\{\frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}, 2\bar{w}\right\} & \Longrightarrow \bar{t}^{after} = \frac{2\bar{F}}{d_{min}} \end{cases}$$

Since $\bar{F} > 0$ and $d_{min} > 0 \Longrightarrow \bar{t}^{after} > \bar{t}^{before}$.

2. <u>Case 2:</u> $\frac{\bar{w}}{\bar{L}} \leq d_{min}$ and $2\bar{w} \leq \min\left\{d_{min}, \frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}\right\}$

$$\begin{cases} \frac{\bar{w}}{\bar{L}} \leq d_{min} & \Longrightarrow \bar{t}^{before} = \frac{\bar{F}\,\bar{L}}{\bar{w}} \\ 2\bar{w} \leq \min\left\{d_{min}, \frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}\right\} & \Longrightarrow \bar{t}^{after} = \frac{\bar{F}}{\bar{w}} \end{cases}$$

It follows that $\left(\bar{t}^{before} > \bar{t}^{after} \Longleftrightarrow \bar{L} > 1\right)$.

3. <u>Case 3:</u> $\frac{\bar{w}}{\bar{L}} \leq d_{min}$ and $\frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}} \leq \min\{d_{min}, 2\bar{w}\}$.

$$\begin{cases} \frac{\bar{w}}{\bar{L}} \leq d_{min} & \Longrightarrow \bar{t}^{before} = \frac{\bar{F}\,\bar{L}}{\bar{w}} \\ \frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}} \leq \min\{d_{min}, 2\bar{w}\} & \Longrightarrow \bar{t}^{after} = \frac{2\bar{F}\,\bar{L}}{\bar{w}+\bar{\eta}\,\bar{L}\,\bar{u}} \end{cases}$$

Thus, $\left(\bar{t}^{before} > \bar{t}^{after} \Longleftrightarrow \bar{\eta}\,\bar{L}\,\bar{u} > \bar{w}\right)$.

4. <u>Case 4:</u> $d_{min} \leq \frac{\bar{w}}{\bar{L}}$ and $d_{min} \leq \min\left\{\frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}, 2\bar{w}\right\}$.

$$\begin{cases} \frac{\bar{w}}{\bar{L}} \leq d_{min} & \Longrightarrow \bar{t}^{before} = \frac{\bar{F}\,\bar{L}}{\bar{w}} \\ d_{min} \leq \min\left\{\frac{\bar{w}+\bar{\eta}\bar{L}\bar{u}}{\bar{L}}, 2\bar{w}\right\} & \Longrightarrow \bar{t}^{after} = \frac{2\bar{F}}{d_{min}} \end{cases}$$

Thus, $\left(\bar{t}^{before} > \bar{t}^{after} \Longleftrightarrow \bar{L}\,d_{min} > 2\,w\right)$.

□

**Summary** Based on this analysis, we can conclude that the efficiency of cross-swarm bundling in data centers depends on the transfer bottleneck. When the data center has plenty of bandwidth to satisfy the demands of the clients, it is not worth bundling. However, when the outgoing bandwidth is scarce, cross-swarm bundling presents a potential solution to reduce the download time for the clients.

## 6.5 Implementing Bundling in Data Centers

In this section, we present the methodology to implement cross-swarm bundling in data centers. We start by calculating the gain in download time that will result from bundling. This gain represents the metric on which the bundling decision will be made. Later, we present a methodology to select the pairs of swarms to bundle based on graph matching techniques. Finally, we present a solution to the security issues that can be associated with cross-swarm bundling.

### 6.5.1 Bundling Metric: Expected Gain in Download Time

The number of requests managed by the data center can be relatively high especially at peak hours. This means that there are numerous bundling choices among the different swarms of clients. This makes the task of choosing the pairs[4] of swarms to be bundled a challenging one. Since the goal of this chapter is to improve the QoE, we consider a bundling strategy based on the *expected gain in download time* that the peers in a given swarm $s_i$ will experience if $s_i$ is bundled with another swarm $s_j$. The idea is to estimate for each pair of swarms the expected gain in download time and based on the obtained values, group the pairs that would benefit the most.

$gain_{s_i}(s_j)$ measures the gain/loss that $s_i$ would experience if it is bundled with $s_j$. This gain can be defined as the normalized ratio of the difference between the expected end download times before and after bundling, as follows:

$$gain_{s_i}(s_j) = \frac{t_{s_i}^{before} - t_{s_i,s_j}^{after}}{t_{s_i}^{before}}. \tag{6.3}$$

$t_{s_i}^{before}$ and $t_{s_i,s_j}^{after}$ are calculated using (2.2) and (4.1), as explained in Section 6.2. $gain_{s_i}(s_j)$ gives an estimation of the gain from the perspective of the swarm $s_i$, while $gain_{s_j}(s_i)$ presents the estimation from the perspective of $s_j$. To complete the picture, we present in (6.4) $gain_{s_i,s_j}$, which represents a weighted aggregation of the gain ratios of both swarms $s_i$ and $s_j$. The weights are allocated based on the number of clients in each swarm ($L_{s_i}$ and $L_{s_j}$).

$$gain_{s_i,s_j} = \frac{L_{s_i}\, gain_{s_i}(s_j) + L_{s_j}\, gain_{s_j}(s_i)}{L_{s_i} + L_{s_j}}. \tag{6.4}$$

[4]To simplify the process of bundle selection, we only consider bundles of two swarms.

$gain_{s_i,s_j}$ is positive if the bundling will result in a notable improvement in the download time for at least one of the swarms. This is the case when both swarms would gain in download time or when the loss of one swarm is lower that the gain of the other. $gain_{s_i,s_j}$ is negative if the bundling will result in a notable degrade in the download time (at least for one swarm). This is the case when both swarms would lose in download time or when the gain of one swarm is lower than the loss of the other. It is important to note here that the gain is symmetric: $gain_{s_i,s_j} = gain_{s_j,s_i}$.

**The objective matrix** We consider now the set of all the swarms managed by the data center and we suppose there is a total of $n$ swarms. To evaluate the efficiency of bundling for each pair of swarms, we calculate the matrix *Gain*. Each row in the matrix corresponds to a swarm $s_i$ and the columns represent the set of potential swarms to be bundled with. Each element $(i, j)$ in *Gain* represents the estimated gain in download time that can result from bundling $s_i$ and $s_j$, which corresponds to $gain_{s_i,s_j}$, calculated using (6.4). Since $gain_{s_i,s_j} = gain_{s_j,s_i}$ and as it makes no sense to bundle the same swarm with itself, we only calculate the gains when $i < j$. Thus, *Gain* is a strictly upper triangular matrix. *Gain* is called the *objective matrix*, and it contains all the expected gains (or losses) in the download times, for all the combinations of bundles. This matrix will serve later to find the optimal set of swarms to bundle.

$$Gain = \begin{pmatrix} 0 & gain_{s_1,s_2} & gain_{s_1,s_3} & \cdots & gain_{s_1,s_n} \\ 0 & 0 & gain_{s_2,s_3} & \cdots & gain_{s_2,s_n} \\ 0 & 0 & 0 & \cdots & gain_{s_3,s_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}. \tag{6.5}$$

### 6.5.2 Finding the Optimal Solution

The number of swarms managed by a data center can be very high. This makes the task of choosing the right combination of swarms to bundle very challenging. In fact, for $n$ different swarms, the total number of possible combinations is: $\binom{n}{2} \times \binom{n-2}{2} \times \binom{n-4}{2} \ldots$.

From this large set of combinations, our goal is to find the "optimal" set of swarm pairs that *maximizes* the total sum of the gains. To do so, we follow the same strategy as Han *et al.* in [63]. We use the *maximum weight matching algorithm* [39, 50] to select the pairs of swarms that should be bundled together by converting the objective matrix $Gain$ into an undirected weighted graph $G = (V, E, w)$, where $w$ is the set of weights that are associated with the edges. Each vertex in $V$ represents a swarm. Each edge in $E$ connecting two swarms (say $s_i$ and $s_j$) signals the possibility of bundling these swarms together. The weight of the edge connecting $s_i$ and $s_j$ is equal to $gain_{s_i,s_j}$.

In graph theory, a *matching* $M$ in $G$ is a subset $E' \subseteq E$ such that no two edges in $E'$ share a common vertex. $M$ is called *maximal*, if it is not a subset of any other matching in $G$. In other words, any extra edge added to $M$ will make $M$ no longer a matching.
With respect to a weighted graph, a *maximum weight matching* is a matching for which the sum of the weights of the matched edges is as large as possible. The first polynomial time algorithm for maximum matching was proposed in 1965 by Edmonds [39] and subsequently improved by Gabow and others [48, 50]. Currently, there are several libraries that can find the maximum weight matchings for dense graphs in time $O(n^3)$, where $n$ is the number of nodes in the original graph.

**Example** Let's consider a simple example where the data center is managing $n = 4$ swarms simultaneously: $S = \{s_1, s_2, s_3, s_4\}$. The total number of combinations of pairs of swarms is: $\binom{4}{2} \times \binom{2}{2} = \frac{4!}{2! \times 2!} \times 1 = 6$. The corresponding objective matrix is:

$$Gain = \begin{pmatrix} 0 & gain_{s_1,s_2} & gain_{s_1,s_3} & gain_{s_1,s_4} \\ 0 & 0 & gain_{s_2,s_3} & gain_{s_2,s_4} \\ 0 & 0 & 0 & gain_{s_3,s_4} \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{6.6}$$

This matrix can be converted into the graph $G$ presented in Figure 6.4. Each node in $G$ represents one of the swarms in the system. Each edge connecting two swarms signals the possibility of bundling these swarms together. The weight of the edge connecting $s_i$ and $s_j$, where $i < j$, is $gain_{s_i,s_j}$.

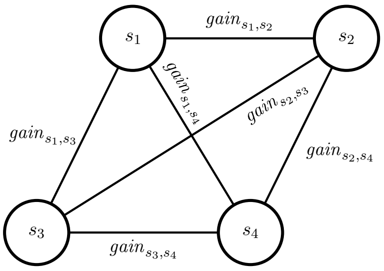


**Figure 6.4:** The graph $G$ resulting from the conversion of the objective matrix $Gain$.

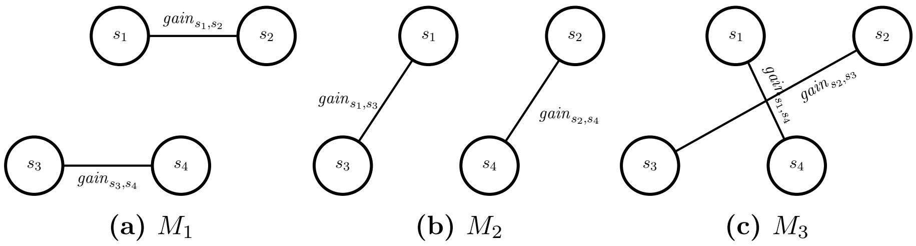


**Figure 6.5:** The three possible matchings of the graph $G$.

From the graph $G$, three possible matchings, $M_1, M_2$ and $M_3$, are possible. These matchings are presented in Figure 6.5. The application of the maximum weight matching algorithm will result in choosing one of these matchings based on the corresponding sum of weights. For instance, if we consider that:
$gain_{s_1,s_2} + gain_{s_3,s_4} \geq \max\left\{gain_{s_1,s_3} + gain_{s_2,s_4}, gain_{s_1,s_4} + gain_{s_2,s_3}\right\}$,
then the result of the maximum weight matching algorithm will be $M_1$. This means that $s_1$ should be bundled with $s_2$ and $s_3$ with $s_4$.

### 6.5.3 Security Concerns

Cross-swarm bundling implies that each swarm will get a copy of a non-requested file along with the requested one. When the files are public, like in the BitTorrent ecosystem, this concept does not pose any security problem. However, in systems like Personal Clouds, where files should be only sent to the authorized entities, cross-swarm bundling can cause a security issue.

To enforce security when the data center decides to bundle two swarms $s_i$ and $s_j$, each of the corresponding files $f_{s_i}$ and $f_{s_j}$ is encrypted separately with a one-shot symmetric key, such as a DES or AES key. We denote by $K_{s_i}$ (respectively $K_{s_j}$) the key used to encrypt $f_{s_i}$ (respectively $f_{s_j}$). After encrypting $f_{s_i}$ and $f_{s_j}$, the data center creates two meta-data `.torrent` files. To ensure content delivery with BitTorrent, both `.torrent` files are sent to the peers in $s_i$ and $s_j$. However, the keys are only sent to the entities authorized to get the content: $K_{s_i}$ is only sent to the clients in $s_i$ and $K_{s_j}$ is only sent to the clients in $s_j$. The keys and the `.torrent` files are sent to each of the requesters via HTTPS.

## 6.6 Application Scenario: Personal Clouds

This section is dedicated to the validation of our bundling strategy. This strategy is validated in Personal Clouds, using the U1 trace described in Chapter 2. The results prove that bundling can improve significantly the QoE without increasing the cloud's bandwidth consumption.

### 6.6.1 Experimental Settings

To evaluate the efficiency of our proposal, we developed a Python[5] simulator that uses the U1 trace and re-simulates the arrival pattern of the clients. The simulator implements the following strategies:

- **HTTP_ONLY:** This strategy represents the default behavior of a Personal Cloud where HTTP is the only download protocol used to distribute the files from the storage servers to the end users.
- **MAX_GAIN:** This scenario corresponds to the application of cross-swarm bundling. At each timestamp, the objective matrix *Gain* is calculated. The choice of swarms to bundle is made based on the application of the maximum weight matching algorithm.
- **MAX_GAIN$>\gamma$:** This scenario is similar to the previous one. However, it adds an extra constraint for the swarms to be bundled in order to avoid taking bundling decisions that could lead to notable degrades in performance. To this extent, the objective matrix *Gain* is

[5]Python Software Foundation: http://www.python.org

filtered before being converted to a graph, and only the combinations $(s_i, s_j)$ that satisfy the condition $gain_{s_i,s_j} > \gamma$ are considered. $\gamma$ is a real value which can be set by system administrator. $\gamma$ can be positive or negative based on the current load on the data center. A negative value of $\gamma$ means that losses of up to $\gamma$ times of the original download time are tolerated. For instance, $\gamma = -1$ means that the bundles considered are the ones that can lead to an increase in download time equal to the original download time at most. Based on our experiments, we have noticed that putting such a limit is very important to prevent degrading the download time for the clients. Note that the *MAX_GAIN* scenario corresponds also to the case *MAX_GAIN*$> \gamma$ where $\gamma = -\infty$.

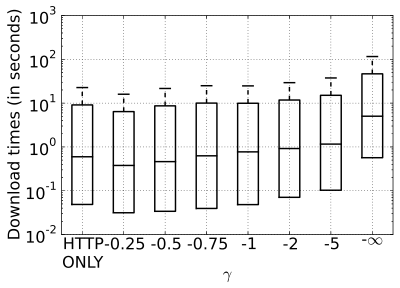
In our implementation of the simulator, we use the NetworkX[6][62] graph matching library. Networkx is based on the "blossom" method for finding augmenting paths and the "primal-dual" method for finding a matching of maximum weight, both methods invented by Edmonds [50].

To evaluate the efficiency of cross-swarm bundling, we propose $\Delta QoE$ as the difference (in seconds) between the download time experienced by a given user downloading a file with HTTP and the download time experienced by the same user downloading the same file after being bundled with another swarm. A positive value of $\Delta QoE$ reflects an improvement in download time compared to the use of HTTP only, while a negative value means that bundling has affected negatively the download time and made it longer. $\Delta QoE = 0$ when there is no change in the download times.

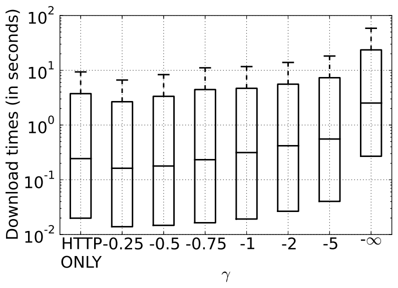
### 6.6.2 Implications of Cross-Swarm Bundling

We exploit the previously described trace sample of U1 and re-simulate the arrival pattern of the peers to validate our approach. We run the simulator with different limits $\gamma \in \{-0.25, -0.5, -0.75, -1, -2, -5, -\infty\}$. We collected the logs of each experiment and evaluated our proposal comparing the results with the ones obtained using HTTP alone.

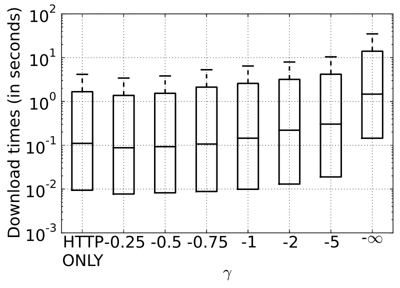
[6]NetworkX is a Python software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Networkx is available under the BSD license at: https://github.com/networkx



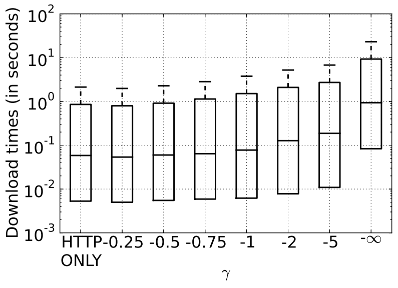
**(a)** $W = 200$ Mbps



**(b)** $W = 250$ Mbps



**(c)** $W = 300$ Mbps

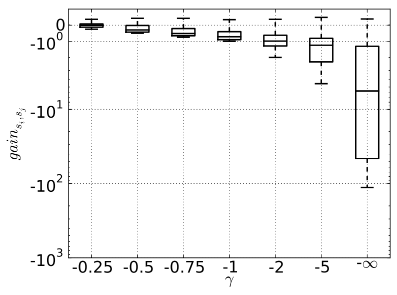


**(d)** $W = 350$ Mbps

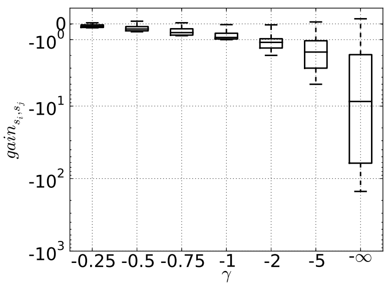
**Figure 6.6:** Measured download times for different cloud bandwidth limits.

Figure 6.6 presents the download times for all the download operations in the trace for different cloud bandwidth limits $W \in \{200, 250, 300, 350$ Mbps$\}$. Compared with the HTTP_ONLY scenario, we notice that bundling performs better under smaller bandwidth limits. Surely, when $W$ gets higher, the download times are shorter and the overall performance of the system improves. However, in these cases, bundling does not improve the download times and sometimes it can even degrade the performance of the system. This result agrees very well with our analysis in Section 6.4.

**Importance of $\gamma$** Figure 6.7 shows the importance of the $\gamma$ constant. It depicts a box plot of the values of the bundling metric $gain_{s_i,s_j}$ for each couple $(s_i, s_j)$ at the moment of bundling. When $\gamma = -\infty$ (MAX_GAIN scenario), the bundling combinations output by the graph matching library



(a) $W = 200$ Mbps



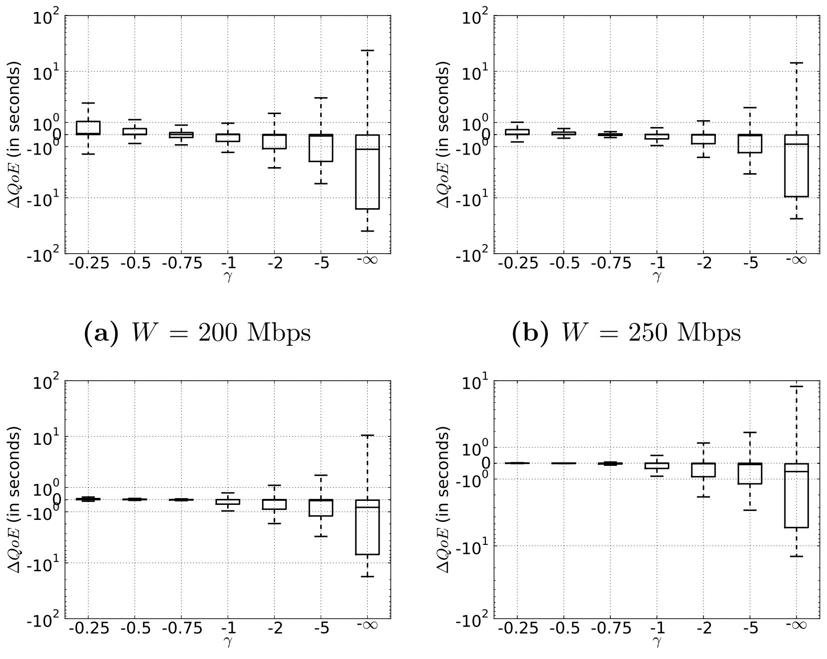
(b) $W = 350$ Mbps

**Figure 6.7:** Values of the bundling metric $gain_{s_i,s_j}$ at the moment of bundling.

can lead to important degradation in download time. In fact, the expected $gain_{s_i,s_j}$ in that scenario can reach -10$^2$ which reflects a resulting download 100 times longer than the original one. This proves that setting $\gamma$ to a reasonable value is very important to prevent degrading the download time for the clients.

This importance can be further perceived in Figure 6.8. This figure presents the $\Delta QoE$ for all the download operations in the trace for different cloud bandwidth limits. $\Delta QoE$ is measured in seconds, and it is calculated for each download operation as the difference between the download time measured with HTTP_ONLY and the one using bundling. We remind that a positive value of $\Delta QoE$ reflects an improvement in download time compared to the use of HTTP alone, while a negative $\Delta QoE$ means that the use of bundling has affected negatively the download time and made it longer. We notice that when bundling was deployed with $\gamma = -\infty$, most of the clients have experienced an increase in download time of about 10 seconds. Nevertheless, when $\gamma$ is small, the majority of the operations have witnessed an improvement in download time of a few seconds.

**Percentage of bundled swarms** Figure 6.9a presents the percentage of bundled swarms as a function of $\gamma$. For the selected values of $\gamma$, this percentage varies, depending on the cloud's bandwidth budget $W$, between 8% and 94%. We notice that the number of bundled swarms increases with $\gamma$. As a matter of fact, when $\gamma$ is smaller, the bundling combinations



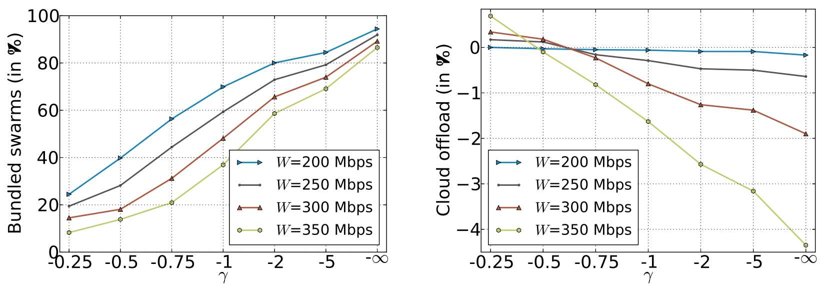
(a) $W$ = 200 Mbps (b) $W$ = 250 Mbps

(c) $W$ = 300 Mbps (d) $W$ = 350 Mbps

**Figure 6.8:** Measured $\Delta QoE$ for different cloud bandwidth limits.

that satisfy the condition $gain_{s_i,s_j} > \gamma$ are more limited which explains the limited percentage of bundled swarms. When $\gamma = -\infty$, the number of bundled swarms is the highest possible (around 90% of the swarms). This is because the application of the bundling algorithm will always result in bundling when there are 2 or more non-bundled swarms.

**Cloud bandwidth usage** In addition to the improvement in download time, the use of bundling does not increase the usage of the cloud's bandwidth a lot. Figure 6.9b depicts the cloud offload percentages for different combinations of $\gamma$ and $W$. This percentage represents the difference between the cloud bandwidth usage with and without bundling. A positive value reflects less bandwidth usage, while a negative value means that bundling requires more bandwidth. In our experiments, the measured



**(a)** Bundled swarms percentage **(b)** Cloud offload percentage

**Figure 6.9:** Percentages of cloud offload and bundled swarms

cloud offload ranges between $-5\%$ and $1\%$. In accordance with the analysis in Section 6.4, we notice savings in cloud bandwidth utilization when the $\Delta QoE$ is positive.

**Performance** In evaluating the performance of cross-swarm bundling, it is equally important to measure the computational overhead needed to select the pairs of swarms to bundle. This overhead is mainly caused by the calculation of the objective matrix and the application of the graph matching library. As mentioned above, in our simulator's implementation, we use the NetworkX [62] library. This library is in Python and it is distributed with the BSD license. As mentioned in the official website[7], NetworkX offers a function that computes a maximum-weighted matching of a graph in $O(n^3)$ time, where $n$ is the number of nodes in a graph.

To quantify this overhead, we measure the delay needed for our simulator to calculate the objective matrix and apply the graph matching function. Figure 6.10 presents this delay for one simulation of the trace ($W = 350$ Mbps and $\gamma = -0.25$). We notice that the execution time grows exceptionally with the number of swarms. Even though we used a regular machine in our simulations, the time needed to select the pairs of swarms to bundle remains reasonable, in the order of a few milliseconds.

[7]`max_weight_matching`: http://networkx.readthedocs.io/en/stable/reference/generated/networkx.algorithms.matching.max_weight_matching.html

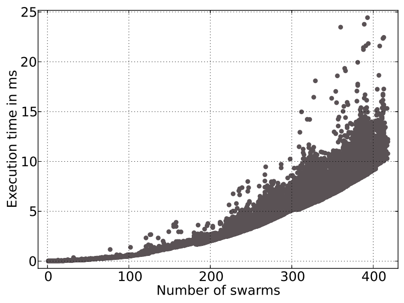


**Figure 6.10:** Overhead related to the calculation of the objective matrix and the application of the graph matching function.

**Summary** Based on the U1 trace, we confirm that *cross-swarm bundling can be useful to improve QoE in Personal Clouds*, especially when the available outgoing bandwidth is small. In this case, cross-swarm bundling can have a positive side effect by reducing the server side bandwidth requirements. The study of the trace has also proven the importance of *filtering the objective matrix before applying graph matching* in order to avoid taking bundling decisions that could lead to degrades in the QoE.

## 6.7 Conclusions

In this chapter, we propose to use cross-swarm bundling in order to improve the QoE for the clients by leveraging their spare upload capacities. To this extent, we present a methodology to implement this feature in Personal Clouds. Our proposal is validated using a real trace of U1. The results prove the efficiency of the approach in improving the QoE for the clients compared to the classic distribution methods.

Bundling can be adopted by Personal Clouds to improve their performance and gain more clients. Our future plans include the study and evaluation of new bundling metrics. We plan also to extend the idea of bundling to systems (other than Personal Clouds) with bigger shared files.

# 7 Conclusions

In this chapter, we conclude our work with a summary of the contributions and results. The chapter contains also a list of the papers published during the dissertation research and some suggestions for future work.

## 7.1 Summary and Findings

In this section, we provide a list of the thesis contributions and a corresponding discussion on the related topics.

### 7.1.1 Summary of contributions

We present here the contributions of this thesis and map them with the three questions raised in Chapter 1. The first question was related to the choice of the most suitable protocol, and it was formulated as follows:

**Question 1** *How to decide which protocol (HTTP or BitTorrent) is more suitable for each file transfer case?*

To answer this question, we started by confuting the general assumption that BitTorrent is only efficient with big files and big swarms and that client-server protocols perform better in the distribution of small files. To this extent, we compared in Chapter 4, the experimental download times with BitTorrent and HTTP. The results proved that, in contrast with general assumptions, BitTorrent can entail better performance than HTTP. The measured improvement in download time reached 22%. Another important advantage of the use of BitTorrent is the amount of data contributed by the peers. This amount is quite important even when the files are small. Starting from just 2 peers, we noticed that the contribution ranged from 11%, for a 1 MB file, to exceed 30%, for a 10 MB file. The peers' contributions was even more important when the number of peers is higher. We also provided in Chapter 4 an accurate estimation of the download time with BitTorrent, optimized for small files/swarms. In this estimation, we reused a fluid model proposed by Kumar *et al.* [75], and extended it to consider the overheads related to the BitTorrent protocol.

In the same context, we proposed two metrics to study the trade-offs between HTTP and BitTorrent. These metrics are the gain and the offload ratios. The gain ratio measures the improvements in terms of download time that can be obtained using BitTorrent instead of HTTP, while the offload ratio measures the amount of data that can be offloaded from the content provider with the use of BitTorrent. These metrics are two key parameters in deciding which download protocol is more suitable for each file transfer case. Using them, we proposed in the same chapter, an algorithm that decides the most suitable protocol and the efficiency of each protocol based on the predefined Quality of Service (QoS) constraints.

The next challenge addressed in this thesis is related to the allocation of the data center's resources, as follows:

**Question 2** *How to balance the available bandwidth resources between the concurrent HTTP and BitTorrent swarms?*

Chapter 5 was dedicated to answering this question. We presented in this chapter a bandwidth allocation algorithm that can be implemented in data centers which are required to serve both HTTP and BitTorrent clients, and which have limited bandwidth resources. Several studies addressed bandwidth allocations in HTTP servers and peer-assisted systems based on BitTorrent separability. But, no related work dealt with both kind of protocols. With our bandwidth allocation algorithm, the data center is able to decide whether to use a client-server approach or a peer-assisted one for each requested file, and to determine the exact amount of bandwidth that should be allocated to that swarm in order to maintain the QoS constraints.

The last problem tackled in this thesis is related to the small number of clients simultaneously interested in the same file. This small number can pose a limit on the maximum benefits of BitTorrent. Thus, it is important to find ways to increase the clients' collaboration. This leads to the third question, which is:

**Question 3** *How to increase the contribution of the clients?*

A good solution to this problem is to "inflate" the swarms of clients by grouping a set of diverse contents into a single `.torrent` file, through cross-swarm bundling. In Chapter 6, we investigated the potential of bundling in a scenario that includes both HTTP and BitTorrent clients. To the best of our knowledge, only bundling of BitTorrent swarms was studied in the literature. In this thesis, we extended this concept and we proposed and evaluated two new bundling variants: the bundling of HTTP swarms, and the bundling of an HTTP swarm with a BitTorrent one. Moreover, we went beyond the classic usage goal of bundling, which is to improve the availability of unpopular torrents, and proposed it as a means to improve the QoE. Actually, the idea of bundling two different swarms in order to improve the QoE might seem controversial at first glance. But, we proved through experimentation and an average case analysis that bundling can effectively improve the QoE, and even reduce the load on the original content provider. Finally, we proposed in the same chapter, a methodology to implement bundling in data centers, based on graph matching techniques.

To summarize, in this thesis, we proposed two different ways to shift part of the burden of file delivery to the end users, through the use of BitTorrent, as follows:

1. **Simultaneous downloads:** BitTorrent can be used between the users requesting the same content at close time intervals. The main idea is to switch the download protocol from HTTP to BitTorrent when there are simultaneous downloads of the same file.

2. **Cross-swarm bundling:** When the number of clients simultaneously interested in the same file is small, the chances of using BitTorrent are limited. Another possible way to make the different clients cooperate is through cross-swarm bundling.

Both techniques were applied in a concrete use case. We selected Personal Clouds as an application scenario since the developers can easily tune the client's implementation to extend it with the BitTorrent functionalities. The results proved that BitTorrent can effectively improve the performance of these systems by reducing the download times and the load on the central servers.

### 7.1.2 Discussion

In this section, we discuss some topics related to the main contributions of this thesis, and which were not discussed in the previous chapters.

**Conservative evaluation** In the evaluation of the benefits of using BitTorrent, we have been conservative in three different aspects: the first is in the measurement of the download times with HTTP, the second is in the management of the peers' resource and the third one is in the choice of the data set.

1. **Conservative in favor of HTTP:** In Chapter 4, we have proposed an estimation of the download time with BitTorrent that takes into consideration the overheads related to the protocol. However, the estimation of the download time with HTTP 2.3 we considered does not take into account the latency related to the protocol headers. Besides, this estimation does not consider the encryption overhead due to the use of HTTPS (which is the transfer protocol used in Personal Clouds in general).
   However, this does not affect negatively the results of our proposals. As a matter of fact, we are being "stricter" on the BitTorrent side. This means that when we include these overheads, the overall improvements resulting from the use of BitTorrent will be even more important.

2. **Conservative in the managements of the peers' resources:** In this thesis, we only consider the worst case scenario when the peers leave the swarm as soon as they finish download. However, this is not always the case, especially in Personal Clouds. In fact, Personal Cloud clients do not disconnect as soon as they finish syncing or downloading a shared file. Instead, they generally continue working on the device and the Personal Cloud program keeps running in the background. To this extent, we believe that if we consider that peers stay in the system after they finish download, the benefits from using BitTorrent will increase significantly.

3. **Conservative with the data set:** The U1 trace used in the validation of the proposed strategies and which presented in Chapter 2, has very limited sharing and involves very small files. To this extent, we believe the benefits of our proposals will be even more important when applied in other systems that involve more sharing and bigger file sizes.

**Mobile clients** An important problem that can be related to the use of BitTorrent on mobile devices, is data cost. In fact, the use of BitTorrent on mobile clients that are on 3G/4G means that clients will upload data, resulting in extra costs on the clients' side. This problem can be easily recovered by limiting the use of the approach to mobile clients that are on WiFi only.

Actually, Personal Clouds try to limit the cellular network usage. In particular, with Dropbox, the files that have been updated or added to the account are only downloaded with a specific request of the user, when the device is on a cellular network. However, on a WiFi network, the updates are automatically downloaded [2].

**Other application scenarios** All our contributions related to the use of BitTorrent in data centers are not limited to Personal Clouds. Actually, they can applied in any scenario, provided that it is possible to extend the clients' implementation to accommodate the BitTorrent protocol. For instance, they can be applied in any web-based application, if the client's web browser is equipped with a BitTorrent extension.

Our contributions can also be applied in *edge computing*. In fact, with the recent technological advances, the frontier of computing applications, data and services is pushed away from centralized servers to the edges of the network [51]. This means that the users can be assimilated to mini-data centers, that can publish and share content all over the Internet. However, the resources of such mini-data centers are still limited, and they can be easily overloaded with requests. In this case, BitTorrent can present an excellent solution to reduce the load on these edges.

*Collaboration technologies* present also a good application example. These systems involve different people working on a common task to achieve their goals, and P2P technologies perform really well in these cases.

## 7.2 Publications

This section gives a list with the papers published during the dissertation research.

### List of Conference Papers

- **Rahma Chaabouni**, Pedro García-López, Marc Sánchez-Artigas, Sandra Ferrer-Celma, and Carlos Cebrian. "*Boosting Content Delivery With BitTorrent in Online Cloud Storage Services.*" In Proceedings of 13th IEEE International Conference on Peer-to-Peer Computing (P2P'13). Trento, Italy. Pages 1-2. *(Demo paper, Core C*[1]*)*

- **Rahma Chaabouni**, Marc Sánchez-Artigas, and Pedro García-López. "*Reducing Costs in the Personal Cloud: Is BitTorrent a Better Bet?*" In Proceedings of 14th IEEE International Conference on Peer-to-Peer Computing (P2P'14). London, UK. Pages 1-10. *(Full paper, Core C*[1]*)*

- **Rahma Chaabouni**, Marc Sánchez-Artigas, Ala Chaabouni, and Pedro García-López. "*Improving the QoE in Personal Clouds with Cross-Swarm Bundling*". In Proceedings of 41st IEEE Conference on Local Computer Networks (LCN'16). Dubai, UAE. Pages 1-9. *(Full paper, Core A*[1]*)*

[1]Source: Computing Research and Education Association of Australasia, CORE, 2014, http://portal.core.edu.au/conf-ranks/

### List of Journal Papers

- Xavier León, **Rahma Chaabouni**, Marc Sánchez-Artigas, and Pedro García López. "*Smart Cloud Seeding for BitTorrent in Datacenters.*" In IEEE Internet Computing, Volume 18, July-Aug. 2014, Pages 47-54. *(Quartile Q1*[2]*)*

- **Rahma Chaabouni**, Marc Sánchez-Artigas, Pedro García-López, and Lluís Pàmies-Juàrez. "*The Power of Swarming in Personal Clouds Under Bandwidth Budget*". In Journal of Network and Computer Applications, Volume 65, April 2016, Pages 48-71. *(Quartile Q1*[2]*)*

- **Rahma Chaabouni**, Marc Sánchez-Artigas, Ala Chaabouni, and Pedro García-López."*Torrentifying Personal Clouds with Cross-Swarm Partial Bundling* ". *(Under submission)*

### Other Contributions

- **Rahma Chaabouni**. "*Reducing Costs in Personal Clouds.*" 1st URV Doctoral Workshop in Computer Science and Mathematics, Llibres URV, Tarragona, Spain, 2014, Pages 5-7.

- **Rahma Chaabouni**. "*Cross-Swarm Bundling in Personal Clouds: Perspectives and Limitations*" 2nd URV Doctoral Workshop in Computer Science and Mathematics, Llibres URV, Tarragona, Spain, 2015, Pages 1-4.

## 7.3 Directions for Future Works

There are several directions to build on the work that we presented in this dissertation. We outline the following ones:

- **New perspectives for BitTorrent:** We have revealed new aspects and domains for the use of BitTorrent. We have proven that this protocol can be efficient in the most adverse scenarios when the file size and the number of peers are very small. With these contributions, we believe we have opened up new perspectives to the use of BitTorrent and extended its domains beyond the classic scenarios to incorporate it in complex systems such as between data centers and

[2]Source: Scimago Journal & Country Rank, 2015, http://www.scimagojr.com/

end users. As stated in the previous section, it is possible to integrate BitTorrent in a variety of use cases, such as: edge computing and collaboration systems.

- **Other bundling metrics:** In Chapter 6, we have proposed a bundling strategy based on the expected gain in download time that the peers in both swarms would experience. However, we believe it is equally important to evaluate the efficiency of bundling using new metrics, such as the amount of data that can be offloaded from the content provider. This could be interesting in a context where the content provider has very limited bandwidth budget and where the peers can tolerate excessive losses in download times. It is even possible to combine both approaches and build new bundling strategies based on both the expected gain in download time and offload in data center's bandwidth.

- **Partial bundling:** Another important proposal is to limit the contribution of the peers in the bundled swarms. Instead of having to download both files entirely, peers in a swarm can only cooperate with part of the other swarm's file. To this extent, it is possible to fix a partial bundle size $\delta$ that represents a minimum limit of the contribution of the peers. Peers who have contributed with $\delta\%$ of the data of the other swarm, can leave the system. This offers more flexibility to the peers and could probably improve the overall benefits that can be driven from cross-swarm bundling.

# Bibliography

[1] Dropbox for Business Security. https://www.dropbox.com/static/business/resources/Security_Whitepaper.pdf.

[2] Dropbox help center: How does syncing work? https://www.dropbox.com/en/help/82.

[3] E. Adar and B. A. Huberman. Free riding on Gnutella, 2000.

[4] K.-M. Ahn and S. Kim. Optimal bandwidth allocation for bandwidth adaptation in wireless multimedia networks . *Computers & Operations Research*, 30(13):1917 – 1929, 2003.

[5] Akamai Technologies, Inc. Akamai Technologies, Inc. https://www.akamai.com/.

[6] Amazon Web Services, Inc. Using BitTorrent with Amazon S3. Available at http://docs.aws.amazon.com/AmazonS3/latest/dev/S3Torrent.html.

[7] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@Home: An Experiment in Public-resource Computing. *Commun. ACM*, 45(11):56–61, Nov. 2002.

[8] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-peer Content Distribution Technologies. *ACM Comput. Surv.*, 36(4):335–371, Dec. 2004.

[9] S. Annapureddy. Providing video-on-demand using peer-to-peer networks. In *In Internet Protocol TeleVision (IPTV) Workshop, WWW 06*, pages 238–247, 2006.

[10] Azureus Software, Inc. Vuze BitTorent client. http://www.vuze.com/.

[11] BitComet Development Group . BitComet. http://www.bitcomet.com/.

[12] BitTorrent, Inc. $\mu$torrent. Light. Limitless. Elegant, efficient torrent downloading. http://www.utorrent.com/.

[13] Box Inc. Box. Your files on any device, from anywhere. https://www.box.com.

[14] H. Bryhni, E. Klovning, and O. Kure. A comparison of load balancing techniques for scalable Web servers. *IEEE Network*, 14(4):58–64, Jul 2000.

[15] A. E. Bryson. Optimal control-1950 to 1985. *IEEE Control Systems*, 16(3):26–33, Jun 1996.

[16] N. Carlsson, D. L. Eager, and A. Mahanti. *Using Torrent Inflation to Efficiently Serve the Long Tail in Peer-AssistedContent Delivery Systems*, pages 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

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

[18] R. Chaabouni, P. Garcia-Lopez, M. Sanchez-Artigas, S. Ferrer-Celma, and C. Cebrian. Boosting Content Delivery with BitTorrent in Online Cloud Storage Services. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–2, Sept 2013.

[19] R. Chaabouni, M. Sanchez Artigas, A. Chaabouni, and P. Garcia Lopez. Improving the QoE in Personal Clouds with Cross-Swarm Bundling. In *Local Computer Networks (LCN), 41st Annual IEEE Conference on*, 2016.

[20] R. Chaabouni, M. Sanchez-Artigas, and P. Garcia-Lopez. Reducing costs in the personal cloud: Is bittorrent a better bet? In *Peer-to-Peer Computing (P2P), 14-th IEEE International Conference on*, pages 1–10, Sept 2014.

[21] R. Chaabouni, M. Sanchez-Artigas, P. Garcia-Lopez, and L. Pamies-Juarez. The Power of Swarming in Personal Clouds Under Bandwidth Budget . *Journal of Network and Computer Applications*, 65:48 – 71, 2016.

[22] A. L. H. Chow, L. Golubchik, and V. Misra. Improving BitTorrent: A Simple Approach. In *Proceedings of the 7th International Conference on Peer-to-peer Systems*, IPTPS'08, pages 8–8, Berkeley, CA, USA, 2008. USENIX Association.

[23] Cisco Systems, Inc. White paper: Cisco VNI Forecast and Methodology, 2015-2020, June 2016.

[24] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[25] C. Dale and J. Liu. A Measurement Study of Piece Population in BitTorrent. In *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, pages 405–410, Nov 2007.

[26] C. Dana, D. Li, D. Harrison, and C. n. Chuah. BASS: BitTorrent Assisted Streaming System for Video-on-Demand. In *2005 IEEE 7th Workshop on Multimedia Signal Processing*, pages 1–4, Oct 2005.

[27] S. Das, S. Tewari, and L. Kleinrock. The Case for Servers in a Peer-to-Peer World. In *2006 IEEE International Conference on Communications*, volume 1, pages 331–336, June 2006.

[28] M. Dee. Inside LAN Sync, October 2015. https://blogs.dropbox.com/tech/2015/10/inside-lan-sync/.

[29] Deluge Team. Deluge BitTorrent Client. http://deluge-torrent.org/.

[30] P. Dhungel, X. Hei, D. Wu, and K. W. Ross. A Measurement Study of Attacks on BitTorrent Seeds. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–5, June 2011.

[31] P. Dhungel, D. Wu, and K. W. Ross. Measurement and mitigation of BitTorrent leecher attacks. *Computer Communications*, 32(17):1852 – 1861, 2009.

[32] P. Dhungel, D. Wu, B. Schonhorst, and K. W. Ross. A Measurement Study of Attacks on BitTorrent Leechers. In *Proceedings of the 7th International Conference on Peer-to-peer Systems*, IPTPS'08, pages 7–7, Berkeley, CA, USA, 2008. USENIX Association.

[33] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing*, 6(5):50–58, Sept. 2002.

[34] M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi. Detecting Bittorrent Blocking. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, pages 3–8, New York, NY, USA, 2008. ACM.

[35] I. Drago. *Understanding and Monitoring Cloud Services*. PhD thesis, University of Twente, 2013.

[36] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. Benchmarking Personal Cloud Storage. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC'13, pages 205–212. ACM, 2013.

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

[38] Dropbox, Inc. https://www.dropbox.com/.

[39] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[40] Ernesto. Dutch University Uses BitTorrent to Update Workstations, March 2008. Available at https://torrentfreak.com/university-uses-utorrent-080306/.

[41] Ernesto. Facebook Uses BitTorrent, and They Love It, June 2010. Available at https://torrentfreak.com/facebook-uses-bittorrent-and-they-love-it-100625/.

[42] J. Farina, M. Scanlon, and M.-T. Kechadi. BitTorrent Sync: First Impressions and Digital Forensic Implications. *Digital Investigation*, 11, Supplement 1:S77 – S86, 2014. Proceedings of the First Annual {DFRWS} Europe.

[43] G. Fedak, H. He, and F. Cappello. BitDew: A data management and distribution service with multi-protocol file transfer and metadata abstraction. *Journal of Network and Computer Applications*, 32(5):961 – 975, 2009. Next Generation Content Networks.

[44] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. https://www.ietf.org/rfc/rfc2616.txt, June 1999.

[45] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. https://www.ietf.org/rfc/rfc7230.txt, June 2014.

[46] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. https://www.ietf.org/rfc/rfc7231.txt, June 2014.

[47] B. Florescu and M. I. Andreica. Towards a Peer-Assisted Content Delivery Architecture. In *Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS) (ISSN: 2066-4451)*, volume 2, pages 521–528, Bucharest, Romania, May 2011.

[48] H. N. Gabow. *Implementations of algorithms for maximum matching on nonbipartite graphs*. PhD thesis, Stanford University, 1973.

[49] L. Gadea. Murder: Fast datacenter code deploys using BitTorrent, July 2010. Available at : https://blog.twitter.com/2010/murder-fast-datacenter-code-deploys-using-bittorrent.

[50] Z. Galil. Efficient Algorithms for Finding Maximum Matching in Graphs. *ACM Comput. Surv.*, 18(1):23–38, Mar. 1986.

[51] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere. Edge-centric computing: Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42, Sept. 2015.

[52] P. Garcia-Lopez, A. Moreno, M. Sanchez-Artigas, M. Vukolic, H. Harkous, T. Papaioannou, H. Zhuang, and S. Langridge. CloudSpaces: Deliverable 2.1 Roadmap of outcomes. http://cloudspaces.eu/deliverables, 2013.

[53] P. Garcia-Lopez, M. Sanchez-Artigas, C. Cotes, G. Guerrero, A. Moreno, and S. Toda. StackSync: Architecturing the Personal Cloud to Be in Sync.

[54] P. Garcia-Lopez, M. Sanchez-Artigas, S. Toda, C. Cotes, and J. Lenton. StackSync: Bringing Elasticity to Dropbox-like File Synchronization. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, pages 49–60. ACM, 2014.

[55] G. Gonalves, I. Drago, A. P. C. da Silva, A. B. Vieira, and J. M. Almeida. The Impact of Content Sharing on Cloud Storage Bandwidth Consumption. *IEEE Internet Computing*, 20(4):26–35, July 2016.

[56] Google Inc. Google Docs. https://www.google.com/docs/about/.

[57] Google Inc. Google Drive. https://www.google.com/intl/en/drive.

[58] Google Inc. Google Sheets. https://www.google.com/sheets/about/.

[59] Google Inc. Google Slides. https://www.google.com/slides/about/.

[60] R. Gracia-Tinedo, Y. Tian, J. Sampe, H. Harkous, J. Lenton, P. Garcia-Lopez, M. Sanchez-Artigas, and M. Vukolic. Dissecting UbuntuOne: Autopsy of a Global-scale Personal Cloud Back-end. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, IMC '15, pages 155–168. ACM, 2015.

[61] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, pages 4–4, Berkeley, CA, USA, 2005. USENIX Association.

[62] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, 2008.

[63] J. Han, T. Chung, S. Kim, H. chul Kim, J. Kangasharju, T. T. Kwon, and Y. Choi. Strategic bundling for content availability and fast distributionin BitTorrent. *Computer Communications*, 43(0):64 – 73, 2014.

[64] J. Han, T. Chung, S. Kim, T. T. Kwon, H.-c. Kim, and Y. Choi. How Prevalent is Content Bundling in BitTorrent. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11, pages 127–128. ACM, 2011.

[65] J. Han, S. Kim, T. Chung, T. T. Kwon, H.-c. Kim, and Y. Choi. Bundling Practice in BitTorrent: What, How, and Why. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 77–88. ACM, 2012.

[66] R. Houdaille and S. Gouache. Shaping HTTP Adaptive Streams for a Better User Experience. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 1–9, New York, NY, USA, 2012. ACM.

[67] C. Huang, J. Li, and K. W. Ross. Peer-Assisted VoD: Making Internet Video Distribution Cheap. The 6th International Workshop on Peer-to-Peer Systems, 02 2007.

[68] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five Months in a torrent's lifetime. In *Passive and Active Network Measurement*, pages 1–11. Springer, 2004.

[69] S. James and P. Crowley. Fast Content Distribution on Datacenter Networks. In *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, ANCS '11, pages 87–88, Washington, DC, USA, 2011. IEEE Computer Society.

[70] S. James and P. Crowley. Experimental analyses of data distribution on data center networks. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sept 2013.

[71] D. Julian, M. Chiang, D. O'Neill, and S. Boyd. QoS and fairness constrained convex optimization of resource allocation for wireless cellular and ad hoc networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 477–486, June 2002.

[72] S. Jun and M. Ahamad. Incentives in BitTorrent Induce Free Riding. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems*, P2PECON '05, pages 116–121, New York, NY, USA, 2005. ACM.

[73] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should Internet Service Providers Fear Peer-assisted Content Distribution? In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, pages 6–6, Berkeley, CA, USA, 2005. USENIX Association.

[74] O. Karsu and A. Morton. Incorporating balance concerns in resource allocation decisions: A bi-criteria modelling approach. *Omega*, 44:70 – 82, 2014.

[75] R. Kumar and K. Ross. Peer-Assisted File Distribution: The Minimum Distribution Time. In *Hot Topics in Web Systems and Technologies, 2006. HOTWEB '06. 1st IEEE Workshop on*, pages 1–11, Nov 2006.

[76] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and Sharing Incentives in BitTorrent Systems. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, pages 301–312, New York, NY, USA, 2007. ACM.

[77] A. Legout, G. Urvoy-Keller, and P. Michiardi. Understanding BitTorrent: An Experimental Perspective. Technical report, I.N.R.I.A. Sophia Antipolis, France, 2005.

[78] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest First and Choke Algorithms Are Enough. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 203–216, New York, NY, USA, 2006. ACM.

[79] N. Lev-tov, N. Carlsson, Z. Li, C. Williamson, and S. Zhang. Dynamic file-selection policies for bundling in BitTorrent-like systems. In *Quality of Service (IWQoS), 2010 18th International Workshop on*, pages 1–9, June 2010.

[80] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. Bittorrent is an Auction: Analyzing and Improving Bittorrent's Incentives. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 243–254, New York, NY, USA, 2008. ACM.

[81] K. Li and S. Jamin. A measurement-based admission-controlled Web server. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 651–659 vol.2, 2000.

[82] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai. Efficient Batched Synchronization in Dropbox-Like Cloud Storage Services. In *Proceedings of the 14th International Middleware Conference*, Middleware '13, pages 307–327, 2013.

[83] J. Liang, R. Kumar, and K. W. Ross. Understanding KaZaA, 2004.

[84] S. Liu, X. Huang, H. Fu, and G. Yang. Understanding Data Characteristics and Access Patterns in a Cloud Storage System. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 327–334, May 2013.

[85] T. S. Loon and V. Bharghavan. Alleviating the Latency and Bandwidth Problems in WWW Browsing. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[86] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A Survey and Comparison of Peer-to-peer Overlay Network Schemes. *Commun. Surveys Tuts.*, 7(2):72–93, Apr. 2005.

[87] D. S. Menasche, A. A. Rocha, B. Li, D. Towsley, and A. Venkataramani. Content Availability and Bundling in Swarming Systems. In *Proceedings of the 5th*

*International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 121–132. ACM, 2009.

[88] R. C. Merton. Lifetime Portfolio Selection under Uncertainty: The Continuous-Time Case. *The Review of Economics and Statistics*, 51(3):247–57, August 1969.

[89] M. Meulpolder, L. D'Acunto, M. Capotă, M. Wojciechowski, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips. Public and Private BitTorrent Communities: A Measurement Study. In *Proceedings of the 9th International Conference on Peer-to-peer Systems*, IPTPS'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[90] P. Michiardi, D. Carra, F. Albanese, and A. Bestavros. Peer-assisted content distribution on a budget . *Computer Networks*, 56(7):2038 – 2048, 2012.

[91] Microsoft Inc. MSN Video. https://www.msn.com/en-us/video.

[92] Microsoft Inc. One Drive. Do more wherever you go. https://onedrive.live.com/.

[93] R. Nikjah and N. C. Beaulieu. Strict suboptimality of selection amplify-and-forward relaying under global channel information. *IEEE Transactions on Communications*, 57(10):2918–2922, October 2009.

[94] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, pages 55–64, New York, NY, USA, 2008. ACM.

[95] A.-M. K. Pathan and R. Buyya. A Taxonomy and Survey of Content Delivery Networks . Technical report, Grid Computing and Distributed Systems Laboratory, University of Melbourne, 2007.

[96] R. S. Peterson and E. G. Sirer. Antfarm: Efficient Content Distribution with Managed Swarms. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, pages 107–122, Berkeley, CA, USA, 2009. USENIX Association.

[97] R. S. Peterson, B. Wong, and E. G. Sirer. A Content Propagation Metric for Efficient Content Distribution. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 326–337, New York, NY, USA, 2011. ACM.

[98] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do Incentives Build Robustness in Bit Torrent. In *Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation*, NSDI'07, pages 1–1, Berkeley, CA, USA, 2007. USENIX Association.

[99] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The BitTorrent P2P File-Sharing System: Measurements and Analysis. In *Peer-to-Peer Systems IV*, pages 205–216. Springer, 2005.

[100] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-like Peer-to-peer Networks. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIG-COMM '04, pages 367–378. ACM, 2004.

[101] A. H. Rasti and R. Rejaie. Understanding Peer-level Performance in BitTorrent: A Measurement Study. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 109–114, Aug 2007.

[102] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh, and D. Rubenstein. VMtorrent: virtual appliances on-demand. In *SIGCOMM*, pages 453–454, 2010.

[103] E. Rescorla. HTTP Over TLS. https://www.ietf.org/rfc/rfc2818.txt, May 2000.

[104] Rhapsody International Inc. Napster. http://us.napster.com/.

[105] I. Rimac, S. Borst, and A. Walid. Peer-assisted content distribution networks: Performance gains and server capacity savings. *Bell Labs Technical Journal*, 13:59 – 69, 2008.

[106] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100, Aug 2001.

[107] S. Saroiu, K. P. Gummadi, and S. D. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Syst.*, 9(2):170–184, Aug. 2003.

[108] M. Scanlon, J. Farina, and M. T. Kechadi. BitTorrent Sync: Network Investigation Methodology. In *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, pages 21–29, Sept 2014.

[109] M. Schmidt, N. Fallenbeck, M. Smith, and B. Freisleben. Efficient Distribution of Virtual Machines for Cloud Computing. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 567–574, 2010.

[110] A. Sharma, A. Venkataramani, and A. A. Rocha. Pros & cons of model-based bandwidth control for client-assisted content delivery. In *2014 Sixth International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–8, Jan 2014.

[111] P. Sinha and A. A. Zoltners. The Multiple-Choice Knapsack Problem. *Operations Research*, 27(3):503–515, 1979.

[112] Statista. Number of personal cloud storage users. http://www.statista.com/statistics/499558/worldwide-personal-cloud-storage-users/.

[113] R. Steinmetz and K. Wehrle. *What Is This "Peer-to-Peer" About?*, pages 9–16. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[114] W. T. Sullivan, III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major SETI project based on Project SERENDIP data and 100,000 personal computers. In C. Batalli Cosmovici, S. Bowyer, and D. Werthimer, editors, *IAU Colloq. 161: Astronomical and Biochemical Origins and the Search for Life in the Universe*, page 729, Jan. 1997.

[115] R. Sweha, V. Ishakian, and A. Bestavros. Angels in the Cloud: A Peer-Assisted Bulk-Synchronous Content Distribution Service. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 97–104, July 2011.

[116] R. Sweha, V. Ishakian, and A. Bestavros. AngelCast: Cloud-based Peer-assisted Live Streaming Using Optimized Multi-tree Construction. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 191–202, New York, NY, USA, 2012. ACM.

[117] The qBittorrent project. qBittorrent. Free and reliable P2P BitTorent client. http://www.qbittorrent.org/.

[118] L. Toka, M. Dell'Amico, and P. Michiardi. Online data backup: A peer-assisted approach. In *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pages 1–10. IEEE, 2010.

[119] Transmission Project. Transmission. A Fast, Easy and Free BitTorent Client. https://transmissionbt.com/.

[120] H. Treasury. Combined Online Information System, June 2010. Available at https://data.gov.uk/dataset/coins.

[121] J. Tyson. How the Old Napster Worked. http://computer.howstuffworks.com/napster2.htm.

[122] J. Wang, C. Huang, and J. Li. On ISP-friendly Rate Allocation for Peer-assisted VoD. In *Proceedings of the 16th ACM International Conference on Multimedia*, MM '08, pages 279–288, New York, NY, USA, 2008. ACM.

[123] L. Wang and J. Kangasharju. Measuring large-scale distributed systems: case of BitTorrent Mainline DHT. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sept 2013.

[124] B. Wei, G. Fedak, and F. Cappello. Scheduling Independent Tasks Sharing Large Data Distributed with BitTorrent. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, pages 219–226, Washington, DC, USA, 2005. IEEE Computer Society.

[125] B. Wei, G. Fedak, and F. Cappello. Towards efficient data distribution on computational desktop grids with BitTorrent . *Future Generation Computer Systems*, 23(8):983 – 989, 2007.

[126] S. J. Wright. *Primal-dual interior-point methods*. Siam, 1997.

[127] C. J. Wu, C. Y. Li, and J. M. Ho. Improving the Download Time of BitTorrent-Like Systems. In *2007 IEEE International Conference on Communications*, pages 1125–1129, June 2007.

[128] N. Zeilemaker, M. Capotă, A. Bakker, and J. Pouwelse. Tribler: P2P Media Search and Sharing. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 739–742, New York, NY, USA, 2011. ACM.

[129] S. Zhang, N. Carlsson, D. Eager, Z. Li, and A. Mahanti. Dynamic File Bundling for Large-scale Content Distribution. In *Proceedings of the 2012 IEEE 37th Conference on Local Computer Networks (LCN 2012)*, LCN '12, pages 601–609. IEEE Computer Society, 2012.

[130] S. Zhang, N. Carlsson, D. L. Eager, Z. Li, and A. Mahanti. Towards a Dynamic File Bundling System for Large-Scale Content Distribution. In *MASCOTS 2011, 19th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Singapore, 25-27 July, 2011*, pages 472–474, 2011.

[131] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, and M. Ponec. Peer-assisted Content Distribution in Akamai Netsession. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 31–42, New York, NY, USA, 2013. ACM.

[132] K. Zhu, D. Niyato, and P. Wang. Optimal Bandwidth Allocation with Dynamic Service Selection in Heterogeneous Wireless Networks. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5, Dec 2010.

# A
# Inverting the Gain's Formulas

The goal of this appendix is to reverse the equations of the gain and get, for a given swarm $s$ and a given file of size $F_s$, the amount of bandwidth needed to be provided by the seed $w_s^*$ that satisfies the condition $Gain_s = \tau$. To emphasize the fact that $w_s^*$ is variable and its value can change significantly the gain ratio, we denote the $Gain_s$ by $Gain_s(w_s^*)$, as follows:

$$Gain_s(w_s^*) = \begin{cases} -\frac{\alpha_{bt}\, d_{min,s}}{F_s}, & \text{if } d_{min,s} \le \frac{w_s^*}{L_s} \text{ and } d_{min,s} \le \min\left\{\frac{w_s^*+\eta_s\, u_s\, L_s}{L_s}, w_s^*\right\} \\ 1 - \frac{w_s^*}{L_s\, d_{min,s}} - \frac{\alpha_{bt}\, w_s^*}{F_s\, L_s}, & \text{if } \frac{w_s^*}{L_s} \le d_{min,s} \text{ and } d_{min,s} \le \min\left\{\frac{w_s^*+\eta_s\, u_s\, L_s}{L_s}, w_s^*\right\} \\ 1 - \frac{w_s^*}{w_s^*+\eta_s\, u_s\, L_s} - \frac{\alpha_{bt}\, w_s^*}{F_s\, L_s}, & \text{if } \frac{w_s^*+\eta_s\, u_s\, L_s}{L_s} \le \min\left\{d_{min,s}, w_s^*\right\} \\ 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, w_s^*}{F_s\, L_s}, & \text{if } w_s \le \min\left\{d_{min,s}, \frac{w_s^*+\eta_s\, u_s\, L_s}{L_s}\right\}. \end{cases}$$

Each of the interval condition is referred to by a case, as in the original paper [75]:

- **Case I:** $d_{min,s} \le \frac{w_s^*}{L_s}$ and $d_{min,s} \le \min\left\{\frac{w_s^*+\eta_s\, u_s\, L_s}{L_s}, w_s^*\right\}$,
- **Case II:** $\frac{w_s^*}{L_s} \le d_{min,s}$ and $d_{min,s} \le \min\left\{\frac{w_s^*+\eta_s\, u_s\, L_s}{L_s}, w_s^*\right\}$,

- **Case II:** $\frac{w_s^* + \eta_s \, u_s \, L_s}{L_s} \leq \min\left\{d_{min,s}, w_s^*\right\}$,
- **Case IV:** $w_s \leq \min\left\{d_{min,s}, \frac{w_s^* + \eta_s \, u_s \, L_s}{L_s}\right\}$.

While inverting this equation and in order to be able to define correctly the interval delimiters, we need to distinguish two different cases based on the maximum of $(L_s - 1)\, d_{min,s}$ and $L_s \, \eta_s \, u_s$:

- **Case A:** $(L_s - 1)\, d_{min,s} \geq L_s \, \eta_s \, u_s$
- **Case B:** $(L_s - 1)\, d_{min,s} \leq L_s \, \eta_s \, u_s$

## Case A: $(L_s - 1)\, d_{min,s} \geq L_s \, \eta_s \, u_s$

The general shape of the $Gain_s(w_s)$ function is given in Figure A.1. In the following sections, we investigate the values o the interval delimiters: $lim_1$, $lim_2$, $lim_3$ and $lim_4$, and the corresponding gain values: $Gain_s(lim_1)$, $Gain_s(lim_2)$, $Gain_s(lim_3)$ and $Gain_s(lim_4)$.

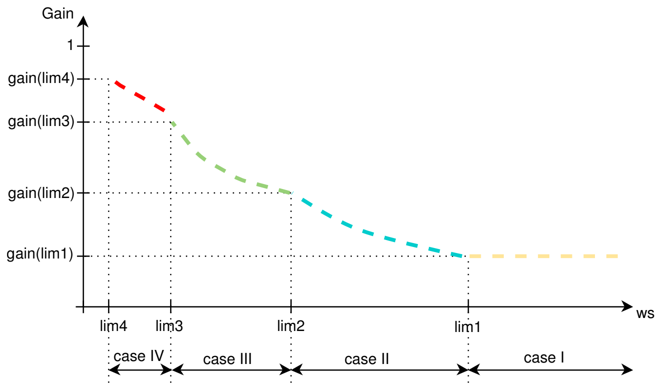


**Figure A.1:** General shape of the gain ratio as a function of the upload speed of the seed for case A when $(L_s - 1)\, d_{min,s} \geq L_s \, \eta_s \, u_s$

***$lim_1$* and *$Gain_s(lim_1)$*** The conditions of **Case I** are the followings:

$$\begin{cases} d_{min,s} \leq \frac{w_s^*}{L_s} \\ d_{min,s} \leq \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \\ d_{min,s} \leq w_s^* \end{cases} \overset{\substack{L_s \geq 1 \\ \eta_s\, u_s \geq 0}}{\Longrightarrow} \begin{cases} d_{min,s} \leq \frac{w_s^*}{L_s} \leq w_s^* \\ d_{min,s} \leq \frac{w_s^*}{L_s} \leq \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \end{cases}$$

$$\Longrightarrow \quad w_s^* \geq L_s\, d_{min,s}$$

$$\Longrightarrow \quad lim_1 = L_s d_{min,s}$$

The corresponding $Gain_s(lim_1)$ for Case I is as follows:

$$Gain_s(lim_1) \overset{caseI}{=} Gain_s(L_s\, d_{min,s}) \overset{caseI}{=} -\frac{\alpha_{bt}.d_{min,s}}{F_s}$$

**Resolution of the equation** $Gain_s(w_s^*) = \tau,\ \forall \tau \in ]-\infty, -\frac{\alpha_{bt}\, d_{min,s}}{F_s}]$

We have $\tau \in ]-\infty, -\frac{\alpha_{bt}.d_{min,s}}{F_s}]$ and $Gain_s(w_s^*) = \tau$.
This leads to $Gain_s(w_s^*) \in ]-\infty, -\frac{\alpha_{bt}\, d_{min,s}}{F_s}]$, thus $w_s^* \geq L_s\, d_{min,s}$.
Thus, $\forall \tau \in ]-\infty, -\frac{\alpha_{bt}\, d_{min,s}}{F_s}]$, the optimal bandwidth that should be allocated to the swarm without violating the constraint $Gain_s(w_s^*) \geq \tau$, is $w_s^* = L_s d_{min,s}$.

$$\boxed{\forall \tau \in \left]-\infty, -\frac{\alpha_{bt}\, d_{min,s}}{F_s}\right],\ \left(Gain_s(w_s^*) = \tau\right) \Rightarrow (w_s^* = L_s\, d_{min,s})}$$

***$lim_2$* and *$Gain_s(lim_2)$*** The conditions of **Case II** are as follows:

$$\begin{cases} d_{min,s} \geq \frac{w_s^*}{L_s} \\ d_{min,s} \leq \frac{w_s^* + \eta._s\, L_s\, u_s}{L_s} \\ d_{min,s} \leq w_s^* \end{cases} \Longrightarrow \begin{cases} w_s^* \geq d_{min,s} \\ w_s^* \geq L_s\,(d_{min,s} - \eta_s\, u_s) \end{cases}$$

$$\Longrightarrow \quad w_s^* \geq \max\left(d_{min,s}, L_s\,(d_{min,s} - \eta_s\, u_s)\right)$$

$$\Longrightarrow \quad lim_2 = \max\left(d_{min,s}, L_s\,(d_{min,s} - \eta_s\, u_s)\right)$$

$$\overset{\text{Case A}}{\Longrightarrow} \quad lim_2 = L_s\,(d_{min,s} - \eta_s\, u_s)$$

Let's verify whether $lim_1 \geq lim_2$:

$$\begin{aligned} lim_1 - lim_2 &= L_s\, d_{min,s} - L_s\,(d_{min,s} - \eta_s\, u_s) \\ &= L_s\, d_{min,s} - L_s\, d_{min,s} + L_s\, \eta_s\, u_s \\ &= L_s\, \eta_s\, u_s \geq 0 \quad (\text{because } L_s \geq 1,\ \eta_s \geq 0 \text{ and } u_s \geq 0) \end{aligned}$$

The corresponding $Gain_s(lim_2)$ for Case II is as follows:

$$\begin{aligned} Gain_s(lim_2) &\overset{caseII}{=} 1 - \frac{L_s\, d_{min,s} - L_s\, \eta_s\, u_s}{L_s\, d_{min,s}} - \frac{\alpha_{bt}(L_s\, d_{min,s} - L_s\, \eta_s\, u_s)}{F_s\, L_s} \\ &= \frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s} \end{aligned}$$

This formula should be verified using the gain formula for Case III since the two cases share the same border $lim_2$:

$$\begin{aligned} Gain_s(lim_2) &\overset{caseIII}{=} 1 - \frac{L_s d_{min,s} - L_s\, \eta_s\, u_s}{L_s d_{min,s} - L_s \eta_s u_s + \eta_s L_s u_s} - \frac{\alpha_{bt}(L_s d_{min,s} - L_s \eta_s u_s)}{F_s\, L_s} \\ &= \frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}\,(d_{min,s} - \eta_s\, u_s)}{F_s} \end{aligned}$$

We need now to verify whether $Gain_s(lim_2) \geq Gain_s(lim_1)$:

$$\begin{aligned} Gain_s(lim_2) - Gain_s(lim_1) &= \frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}\,(d_{min,s} - \eta_s\, u_s)}{F_s} + \frac{\alpha_{bt}\, d_{min,s}}{F_s} \\ &= \frac{\eta_s\, u_s}{d_{min,s}} + \frac{\alpha_{bt}\, \eta_s\, u_s}{F_s} \geq 0 \end{aligned}$$

$$\text{Thus,} \quad Gain_s(lim_2) \geq Gain_s(lim_1)$$

**Resolution of the equation** $Gain_s(w_s^*) = \tau, \ \forall \tau \in \left[ -\frac{\alpha_{bt}.d_{min,s}}{F_s}, \frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s} \right]$

We have $Gain_s(w_s^*) = \tau$ and $\tau \in \left[ -\frac{\alpha_{bt}\, d_{min,s}}{F_s}, \frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s} \right]$
This means that $Gain_s(w_s^*) \in \left[ -\frac{\alpha_{bt}\, d_{min,s}}{F_s}, \frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s} \right]$ which corresponds to the formula of the gain related to Case II.
Let's try now to invert that formula in order to get an estimation of $w_s^*$:

$$\begin{aligned} Gain_s(w_s^*) = \tau \quad &\overset{\text{case II}}{\Longleftrightarrow} \quad 1 - \frac{w_s^*}{L_s.d_{min,s}} - \frac{\alpha_{bt}.w_s^*}{F_s\, L_s} = \tau \\ &\Longleftrightarrow \quad 1 - \tau = w_s^* \left( \frac{1}{L_s\, d_{min,s}} + \frac{\alpha_{bt}}{F_s\, L_s} \right) \\ &\Longleftrightarrow \quad w_s^* = \frac{(1-\tau)\, F_s\, L_s\, d_{min,s}}{F_s + d_{min,s}\, \alpha_{bt}} \end{aligned}$$

We can then conclude that:

$$\boxed{\begin{aligned} &\forall \tau \in \left[ -\frac{\alpha_{bt}\, d_{min,s}}{F_s}, \frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s} \right], \\ &\qquad \Big( Gain_s(w_s^*) = \tau \Big) \Rightarrow w_s^* = \frac{(1-\tau)\, F_s\, L_s\, d_{min,s}}{F + d_{min,s}\, \alpha_{bt}} \end{aligned}}$$

***$lim_3$* and *$Gain_s(lim_3)$*** The conditions of **Case III** are the followings:

$$\begin{cases} \frac{w_s^*+\eta_s\,L_s\,u_s}{L_s} \leq d_{min,s} \\ \frac{w_s^*+\eta_s\,L_s\,u_s}{L_s} \leq w_s^* \end{cases} \implies \begin{cases} w_s^* \leq L_s(d_{min,s} - \eta_s\,u_s) \\ w_s^* \geq \frac{\eta_s\,L_s\,u_q}{L_s-1} \end{cases}$$
$$\implies lim_3 = \frac{\eta_s\,L_s\,u_s}{L_s-1}$$

Let's verify whether $lim_2 \geq lim_3$:

$$\begin{aligned} lim_2 - lim_3 &= L_s\,(d_{min,s} - \eta_s\,u_s) - \frac{\eta_s\,L_s\,u_s}{L_s-1} \\ &= \frac{L_s}{L_s-1}\,(L_s\,(d_{min,s} - \eta_s\,u_s) - d_{min,s}) \\ &\geq 0 \quad \text{(because } L_s > 1 \text{ and } d_{min,s} \leq L_s\,(d_{min,s} - \eta_s\,u_s)) \end{aligned}$$

The corresponding $Gain_s(lim_3)$ for Case III is:

$$\begin{aligned} Gain_s(lim_3) &\overset{caseIII}{=} Gain_s\left(\frac{\eta_s\,L_s\,u_s}{L_s-1}\right) \\ &\overset{caseIII}{=} 1 - \frac{\frac{\eta_s\,L_s\,u_s}{L_s-1}}{\frac{\eta_s\,L_s\,u_s}{L_s-1} + \eta_s\,L_s\,u_s} - \frac{\alpha_{bt}\,\frac{\eta_s L_s\,u_s}{L_s-1}}{F_s\,L_s} \\ &= 1 - \frac{\eta_s\,L_s\,u_s}{\eta_s\,L_s\,u_s + \eta_s\,L_s\,u_s\,(L_s-1)} - \frac{\alpha_{bt}\,\eta_s\,u_s}{(L_s-1)\,F_s} \\ &= 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\,u_s}{(L_s-1)\,F_s} \end{aligned}$$

This expression needs to be verified also using the gain formula for case IV since the two cases share the same border $lim_3$:

$$\begin{aligned} Gain_s(lim_3) &\overset{caseIV}{=} Gain_s\left(\frac{\eta_s\,L_s\,u_s}{L_s-1}\right) \overset{caseIV}{=} 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\left(\frac{\eta_s\,L_s\,u_s}{L_s-1}\right)}{F_s\,L_s} \\ &= 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\,u_s}{(L_s-1)\,F_s} \end{aligned}$$

We need now to verify whether $Gain_s(lim_3) \geq Gain_s(lim_2)$:

$$\begin{aligned} Gain_s(lim_3) &- Gain_s(lim_2) \\ &= 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\,u_s}{(L_s-1)\,F_s} - \frac{\eta_s\,u_s}{d_{min,s}} + \frac{\alpha_{bt}(d_{min,s} - \eta_s\,u_s)}{F_s} \\ &= \left[1 - \frac{1}{L_s} - \frac{\eta_s\,u_s}{d_{min,s}}\right] + \left[\frac{\alpha_{bt}\,d_{min,s}}{F_s} - \frac{\alpha_{bt}\,\eta_s\,u_s\,L_s}{F_s\,(L_s-1)}\right] \\ &= \left[\frac{1}{L_s d_{min,s}} + \frac{\alpha_{bt}}{F_s(L_s-1)}\right](L_s(d_{min,s} - \eta_s u_s) - d_{min,s}) \\ &\geq 0 \qquad \text{(since } L_s > 1 \text{ and } d_{min,s} \leq L_s(d_{min,s} - \eta_s\,u_s)\text{ )} \end{aligned}$$

$$\text{Thus,} \qquad Gain_s(lim_3) \geq Gain_s(lim_2)$$

**Resolution of the equation** $Gain_s(w_s^*) = \tau,\ \forall \tau \in \left[\frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\, u_s}{(L_s - 1)\, F_s}\right]$

We have: $\begin{cases} Gain_s(w_s^*) = \tau \\ \tau \in \left[\frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\, u_s}{(L_s - 1)\, F_s}\right] \end{cases}$

This means that:

$$Gain_s(w_s^*) \in \left[\frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\, u_s}{(L_s - 1)\, F_s}\right]$$

which corresponds to the formula of the gain related to Case III.
Let's try now to invert that formula in order to get an estimation of $w_s^*$.

$$Gain_s(w_s^*) = \tau \quad \overset{\text{case III}}{\Longleftrightarrow} \quad 1 - \frac{w_s^*}{w_s^* + \eta_s\, L_s\, u_s} - \frac{\alpha_{bt}\, w_s^*}{F_s\, L_s} = \tau$$

Since the resolution of this equation is quite complex, we can simplify it by introducing the following symbols: $a = \eta_s\, L_s\, u_s$, $b = \frac{\alpha_{bt}}{F_s\, L_s}$ and $c = \tau$. The simplified equation becomes:

$$1 - \frac{w_s^*}{w_s^* + a} - b\, w_s^* = c$$

To solve this second degree equation, we use an online solver [1] and obtain the following solutions:

$$\begin{aligned} w_{s_1}^* &= \frac{\sqrt{a^2\, b^2 - 2\, a\, b\, c + 4\, a\, b + c^2} - a\, b - c}{2\, b} \\ w_{s_2^*} &= \frac{-\left(\sqrt{a^2\, b^2 - 2\, a\, b\, c + 4\, a\, b + c^2} + a\, b + c\right)}{2\, b} \end{aligned}$$

Clearly, $w_{s_2}^* < 0$, so it cannot be considered as a solution. Then, we can conclude that:

$$\boxed{\begin{aligned} &\forall \tau \in \left[\frac{\eta_s\, u_s}{d_{min,s}} - \frac{\alpha_{bt}(d_{min,s} - \eta_s\, u_s)}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\, u_s}{(L_s - 1)\, F_s}\right], \\ &\qquad\qquad \Big(Gain_s(w_s^*) = \tau\Big) \Longrightarrow w_s^* = \frac{\sqrt{a^2\, b^2 - 2\, a\, b\, c + 4\, a\, b + c^2} - a\, b - c}{2\, b} \\ &\qquad \text{where } a = \eta_s\, L_s\, u_s, b = \frac{\alpha_{bt}}{F_s\, L_s}, \text{ and } c = \tau \end{aligned}}$$

[1] The online solver is available at: http://www.wolframalpha.com

***$lim_4$* and *$Gain_s(lim_4)$*** The conditions of Case IV are as follows:

$$\begin{cases} w_s^* \leq \frac{w_s^*+\eta_s\,L_s\,u_s}{L_s} \\ w_s^* \leq d_{min,s} \end{cases} \Longrightarrow \begin{cases} w_s^* \leq \frac{\eta_s\,L_s\,u}{L_s-1} & (lim_2) \\ w_s^* \leq d_{min,s} \end{cases}$$

We need compare $d_{min,s}$ and $\frac{\eta_s\,L_s\,u_s}{L_s-1}$ in order to verify $lim_3$.

$$\begin{aligned} d_{min,s} - \frac{\eta_s L_s u_s}{L_s-1} &= \frac{(L_s-1)\,d_{min,s}-\eta_s\,L_s\,u_s}{L_s-1} = \frac{L_s\,(d_{min,s}-\eta_s\,u_s)-d_{min,s}}{L_s-1} \\ &\geq 0 \quad (\text{because } L_s > 1 \;\&\; d_{min,s} \leq L_s(d_{min,s}-\eta_s u_s)) \end{aligned}$$

Thus $lim_3$'s definition is correct and there no analytic definition for $lim_4$. We can suppose that it can be equal to 0 since $w_s^*$ can only be positive (or equal to 0). So, we can suppose that $lim_4 = 0$, even thought attaining that limit means that the download might be interrupted.

We need now to calculate $\lim_{w_s^* \to 0} Gain_s(w_s^*)$ that will be considered the upper bound of the gain values:

$$\lim_{w_s^* \to 0} (Gain_s(w_s^*)) \overset{caseIV}{=} \lim_{w_s^* \to 0} \left(1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,w_s^*}{F\,L_s}\right) = 1 - \frac{1}{L_s}$$

**Resolution of the equation** $Gain_s(w_s^*) = \tau, \forall \tau \in \left[1 - \frac{1}{L_s} - \frac{\alpha_{bt}\eta_s u_s}{(L_s-1)\,F_s},\right.$ $\left.1 - \frac{1}{L_s}\right[$

We have: $Gain_s(w_s^*) = \tau$ and $\tau \in \left[1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\,u_s}{(L_s-1)\,F_s}, 1 - \frac{1}{L_s}\right[$

This means that $Gain_s(w_s^*) \in \left[1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\,u_s}{(L_s-1)\,F_s}, 1 - \frac{1}{L_s}\right[$, which corresponds to the formula of the gain related to Case IV. Let's try now to invert that formula in order to get an estimation of $w_s$:

$$\begin{aligned} Gain_s(w_s^*) = \tau \quad &\overset{\text{case IV}}{\Longleftrightarrow} \quad 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,w_s^*}{F_s\,L_s} = \tau \\ &\Longleftrightarrow \quad \frac{\alpha_{bt}\,w_s^*}{F_s\,L_s} = 1 - \frac{1}{L_s} - \tau \\ &\Longleftrightarrow \quad w_s^* = \frac{F_s\,L_s}{\alpha_{bt}} \left(\frac{L_s\,(1-\tau)-1}{L_s}\right) \\ &\Longleftrightarrow \quad w_s^* = \frac{F_s\,[L_s\,(1-\tau)-1]}{\alpha_{bt}} \end{aligned}$$

We can then conclude that:

$$\boxed{\begin{aligned}&\forall \tau \in \left[1-\tfrac{1}{L_s}-\tfrac{\alpha_{bt}\,\eta\,u}{(L_s-1)\,F_s},1-\tfrac{1}{L_s}\right[,\\ &\qquad\qquad\left(Gain_s(w_s^*)=\tau\right)\Rightarrow w_s^*=\tfrac{F_s\,[L_s\,(1-\tau)-1]}{\alpha_{bt}}\end{aligned}}$$

**General conclusion for Case A:** The equation: $Gain_s(w_s^*)=\tau$ has the following solution:

$$w_s^*=\begin{cases} L_s\,d_{min,s}, & \forall\tau\in\left]-\infty,-\dfrac{\alpha_{bt}d_{min,s}}{F_s}\right] \\ \dfrac{(1-\tau)F_s\,L_s\,d_{min,s}}{F_s+d_{min,s}\,\alpha_{bt}}, & \forall\tau\in\left[-\dfrac{\alpha_{bt}\,d_{min,s}}{F_s},\dfrac{\eta_s\,u_s}{d_{min,s}}-\dfrac{\alpha_{bt}\,(d_{min,s}-\eta_s\,u_s)}{F_s}\right] \\ \dfrac{\sqrt{a^2b^2-2abc+4ab+c^2}-ab-c}{2\,b}, & \forall\tau\in\left[\dfrac{\eta_s\,u_s}{d_{min,s}}-\dfrac{\alpha_{bt}(d_{min,s}-\eta_s\,u_s)}{F_s},1-\dfrac{1}{L_s}-\dfrac{\alpha_{bt}\,\eta_s\,u_s}{(L_s-1)\,F_s}\right] \\ \dfrac{F_s\,[L_s\,(1-\tau)-1]}{\alpha_{bt}}, & \forall\tau\in\left[1-\dfrac{1}{L_s}-\dfrac{\alpha_{bt}\,\eta_s\,u_s}{(L_s-1)\,F_s},1-\dfrac{1}{L_s}\right[ \\ \nexists, & \forall\tau\in\left[1-\dfrac{1}{L_s},+\infty\right[ \end{cases}$$

Where: $a=\eta_s\,L_s\,u_s$, $b=\frac{\alpha_{bt}}{F_s\,L_s}$ and $c=\tau$.

## Case B: $(L_s-1)\,d_{min,s}\leq L_s\,\eta_s\,u_s$

In this case, the intervals and the delimiters are not as evident as in Case A. So let's start first by studying the limits in the gain cases and try to locate the cases based on their order.

### Fixing the interval delimiters

For each interval, we part from its constraints and determine the range of values of $w_s^*$ in each of these intervals. The goal is to verify if theses intervals are disjoint and do not superimpose.

1. ***lim*₁** (Case I)

$$\begin{cases} d_{min,s}\leq\frac{w_s^*}{L_s} \\ d_{min,s}\leq\frac{w_s^*+\eta_s\,L_s\,u_s}{L_s} \\ d_{min,s}\leq w_s^* \end{cases} \implies d_{min,s}\leq\frac{w_s^*}{L_s}$$

$$\implies w_s^*\geq lim_1=L_s\,d_{min,s}$$

2. $\boldsymbol{lim_2}$ (Case II)

$$\begin{cases} d_{min,s} \geq \frac{w_s^*}{L_s} \\ d_{min,s} \leq \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \\ d_{min,s} \leq w_s^* \end{cases} \Longrightarrow \begin{cases} w_s^* \leq L_s\, d_{min,s} = lim_1 \\ w_s^* \geq d_{min,s} \\ w_s^* \geq L_s(d_{min,s} - \eta_s\, u_s) \end{cases}$$

$$\overset{\text{Case B}}{\Longrightarrow} \begin{cases} w_s^* \leq lim_1 \\ w_s^* \geq d_{min,s} = lim_2 \end{cases}$$

Comparing $lim_1$ and $lim_2$: We know that $L_s > 1$ and $d_{min,s} > 0$. Thus $L_s\, d_{min,s} > d_{min,s} \Rightarrow lim_1 > lim_2$

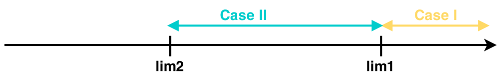


**Figure A.2:** Delimiter $lim_2$ and intervals for Case II

3. $\boldsymbol{lim_3}$ and $\boldsymbol{lim_{2p}}$ (Case III)

$$\begin{cases} \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq d_{min,s} \\ \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq w_s^* \end{cases} \Rightarrow \begin{cases} w_s^* \leq L_s(d_{min,s} - \eta_s u_s) = lim_{2p} \\ w_s^* \geq \frac{\eta\, L_s\, u_s}{L_s - 1} = lim_3 \end{cases}$$

**Comparing $lim_{2p}$ and $lim_2$:** Based on the conditions of Case B, we already know that $lim_{2p} \leq lim_2$

**Comparing $lim_3$ and $lim_2$:**

$$\begin{aligned} lim_3 - lim_2 &= \frac{\eta_s\, L_s\, u_s}{L_s - 1} - d_{min,s} \\ &= \frac{\eta_s\, L_s\, u_s - (L_s - 1)\, d_{min,s}}{L_s - 1} \\ &= \frac{1}{L_s - 1}\left(d_{min,s} - L_s\left(d_{min,s} - \eta_s\, u_s\right)\right) \\ &\geq 0 \text{ (because } L_s > 1 \text{ and } d_{min,s} \geq L_s(d_{min,s} - \eta_s\, u_s)) \\ &\Rightarrow lim_3 \geq lim_2 \end{aligned}$$

**Comparing $lim_3$ and $lim_1$:**

$$\begin{aligned}
lim_3 - lim_1 &= \frac{\eta_s\, L_s\, u_s}{L_s-1} - L_s\, d_{min,s} = \frac{\eta_s\, L_s\, u_s - L_s(L_s-1).d_{min,s}}{L_s-1} \\
&= \frac{\left(L_s\,(d_{min,s}-\eta_s\, u_s) - L_s^2\, d_{min,s}\right)}{L_s-1} \\
\text{Since} \quad & \begin{cases} d_{min,s} \geq L_s(d_{min,s} - \eta_s\, u_s) \\ L_s^2\, d_{min,s} \geq d_{min,s} \end{cases} \\
&\Longrightarrow L_s^2\, d_{min,s} \geq L_s(d_{min,s} - \eta_s\, u_s) \\
\text{Thus} \quad & lim_3 - lim_1 \leq 0 \Longrightarrow lim_1 \geq lim_3
\end{aligned}$$

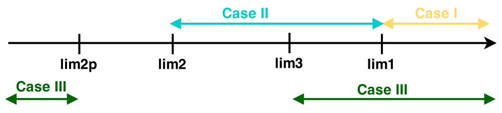


**Figure A.3:** Delimiter $lim_{2p}$ and $lim_3$ and intervals for Case III

4. $\boldsymbol{lim_4}$: (Case IV)

$$\begin{cases} w_s^* \leq \frac{w_s+\eta_s\, L_s\, u_s}{L_s} \\ w_s^* \leq d_{min,s} \end{cases} \Longrightarrow \begin{cases} w_s^* \leq \frac{\eta_s\, L_s\, u_s}{L_s-1} \\ w_s^* \leq d_{min,s} \end{cases}$$

$$\Longrightarrow \begin{cases} w_s^* \leq lim_3 \\ w_s^* \leq lim_2 \end{cases}$$

$$\overset{lim_3 \geq lim_2}{\Longrightarrow} \quad w_s^* \leq lim_2$$

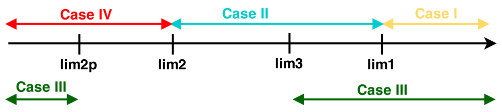


**Figure A.4:** Intervals for Case IV

## Interpretation of the superimposed cases

Based on Figure A.4, we can distinguish 3 intervals where there are superimposed cases which are:

- **Interval 1:** $]-\infty, lim_{2p}] = ]-\infty, L_s\,(d_{min,s} - \eta_s\, u_s)]$ : superimposition of Cases IV and III,
- **Interval 2:** $[lim_3, lim_1] = \left[\frac{\eta_s\, L_s\, u_s}{L_s-1}, L_s\, d_{min,s}\right]$ : superimposition of Cases II and III,
- **Interval 3:** $[lim_1, +\infty[ = [L_s\, d_{min,s}, +\infty[$ : superimposition of Cases I and III.

Let's check interval by interval the implications of such a superimposition. For each interval we will define the new constraints resulting from the intersection of the corresponding cases and define accordingly the gain expression. The goal is to demonstrate that the solution will be the same for both cases.

1. **Interval 1:**

$$\begin{cases} \text{Case IV} : \begin{cases} w_s^* \leq d_{min,s} \\ w_s^* \leq \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \end{cases} \\ \text{Case III} : \begin{cases} \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq d_{min,s} \\ \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq w_s^* \end{cases} \end{cases} \Longrightarrow w_s^* = \frac{L_s\, \eta_s\, u_s}{L_s-1}$$

**Verification of the gain expression in both cases:** Let's now verify that $Gain(w_s, s)$ has the same expression in both cases IV and III when $w_s^* = \frac{\eta_s\, L_s\, u_s}{L_s-1}$.

$$Gain_s\left(\frac{L_s\,\eta_s\,u_s}{L_s-1}\right) \overset{\text{case IV}}{=} 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\frac{\eta_s\,L_s\,u_s}{L_s-1}}{F_s\,L_s} = 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,\eta_s\,u_s}{F_s\,(L_s-1)}$$

$$\begin{aligned} Gain_s\left(\frac{\eta_s\,L_s\,u_s}{L_s-1}\right) &\overset{\text{case III}}{=} 1 - \frac{\frac{\eta_s\,L_s\,u_s}{L_s-1}}{\frac{\eta_s\,L_s\,u_s}{L_s-1} + \eta_s\,L_s\,u_s} - \frac{\alpha_{bt}\,\frac{\eta_s\,L_s\,u_s}{L_s-1}}{F_s\,L_s} \\ &= 1 - \frac{1}{1+(L_s-1)} - \frac{\alpha_{bt}.\eta_s\,u_s}{F_s\,(L_s-1)} \\ &= 1 - \frac{1}{L_s} - \frac{\alpha_{bt}.\eta_s\,u_s}{F_s\,(L_s-1)} \end{aligned}$$

For this interval, both cases have the same expression. Thus, we can just consider just one of the cases instead of working with both. The best choice seems to be Case IV since it has a simpler formulas.

2. **Interval 2:**

$$\begin{cases} \text{Case II}: \begin{cases} d_{min,s} \geq \frac{w_s^*}{L_s} \\ d_{min,s} \leq \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \\ d_{min,s} \leq w_s^* \end{cases} \\ \text{Case III}: \begin{cases} \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq d_{min,s} \\ \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq w_s^* \end{cases} \end{cases} \Longrightarrow w_s^* = L_s\ (d_{min,s} - \eta_s\, u_s)$$

**Verification of the gain expression in both cases:** Let's now verify that $Gain_s(w_s^*)$ has the same expression in both Cases II and III, when $w_s^* = L_s\, (d_{min,s} - \eta_s\, u_s)$.

$$\begin{aligned} Gain_s(L_s d_{min,s}) &\overset{\text{case II}}{=} 1 - \frac{L_s\,(d_{min,s} - \eta_s\, u_s)}{L_s d_{min,s}} - \frac{\alpha_{bt}\, L_s\,(d_{min,s} - \eta_s\, u_s)}{F_s\, L_s} \\ Gain_s(L_s d_{min,s}) &\overset{\text{case III}}{=} 1 - \frac{L_s(d_{min,s} - \eta_s u_s)}{L_s(d_{min,s} - \eta_s u_s) + \eta_s L_s u_s} - \\ &\qquad \frac{\alpha_{bt} L_s(d_{min,s} - \eta_s u_s)}{F_s\, L_s} \\ &= 1 - \frac{L_s(d_{min,s} - \eta_s u_s)}{L_s\, d_{min,s}} - \frac{\alpha_{bt}\, L_s\,(d_{min,s} - \eta_s\, u_s)}{F_s\, L_s} \end{aligned}$$

For this interval, both cases have the same expression. Thus, we can just consider just one of the cases instead of working with both. The best choice seems to be Case II since it has a simpler formulas.

3. **Interval 3:**

$$\begin{cases} \text{Case I}: \begin{cases} d_{min,s} \leq \frac{w_s^*}{L_s} \\ d_{min,s} \leq \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \\ d_{min,s} \leq w_s^* \end{cases} \\ \text{Case III}: \begin{cases} \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq d_{min,s} \\ \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq w_s^* \end{cases} \end{cases} \overset{L_s \geq 1}{\Longrightarrow} \quad d_{min,s} = \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \leq \frac{w_s^*}{L_s}$$

$$\overset{\substack{L_s \geq 1 \\ \eta_s u_s \geq 0}}{\Longrightarrow} \begin{cases} d_{min,s} = \frac{w_s^* + \eta_s\, L_s\, u_s}{L_s} \\ \eta_s\, L_s\, u_s = 0 \end{cases} \Longrightarrow \qquad w_s^* = L_s\, d_{min,s}$$

**Verification of the gain expression in both cases:** Let's now erify that $Gain_s(w_s^*)$ has the same expression in both Cases II and III when $w_s^* = L_s\, d_{min,s}$.

$$\begin{aligned} Gain_s(L_s.d_{min,s}) &\overset{\text{case III}}{=} 1 - \frac{L_s\, d_{min,s}}{L_s\, d_{min,s} + \eta_s\, L_s\, u_s} - \frac{\alpha_{bt}\, L_s\, d_{min,s}}{F_s\, L_s} \\ &\overset{\eta_s\, u_s = 0}{=} 1 - 1 - \frac{\alpha_{bt}\, d_{min,s}}{F_s} \\ &= -\frac{\alpha_{bt}\, d_{min,s}}{F_s} \\ &\overset{\text{case I}}{=} Gain_s(L_s\, d_{min,s}) \end{aligned}$$

For this interval, both cases have the same expression. Thus, we can consider just one of the cases instead of working with both. The best choice seems to be Case I since it has a simpler formulas.

After the verification of the superimposed intervals, we can just consider the following intervals (Figure A.5).

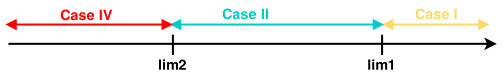


**Figure A.5:** Final intervals to be considered for Case B

## Inverting the gain

The goal of this section is to derive a potential equation of the gain ratio as a function of the seed's upload speed. The general shape of that function is given in Figure A.1. We have already defined the intervals delimiters: $lim_1$ and $lim_2$ and $lim_3$. We just need to verify the corresponding gain values: $Gain_s(lim_1)$, $Gain_s(lim_2)$ and $Gain_s(lim_3)$ that should be equal to the ones already defined in Case A.

$\boldsymbol{lim_1}$ **and** $\boldsymbol{Gain_s(lim_1)}$ We have already found that $lim_1 = L_s\, d_{min,s}$. The corresponding gain has been already calculated and verified for both Case I and Case II. It is equal to: $Gain_s(lim_1) = -\frac{\alpha_{bt}\, d_{min,s}}{F_s}$.

**Resolution of the equation** $Gain_s(w_s^*) = \tau, \forall \tau \in \left]-\infty, -\frac{\alpha_{bt} d_{min,s}}{F_s}\right]$

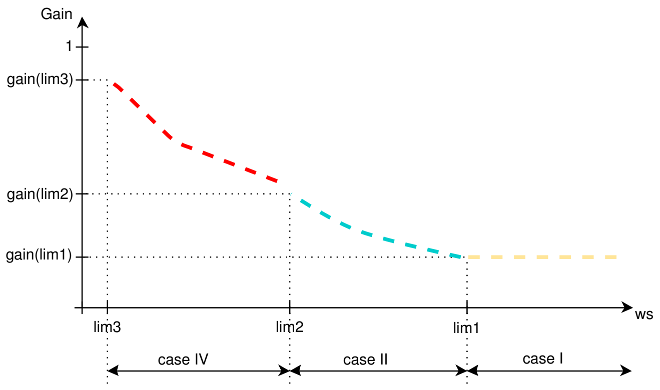


**Figure A.6:** General shape of the gain ratio as a function of the upload speed of the seed for Case B when $\max(d_{min,s}, L_s\,(d_{min,s} - \eta_s\,u_s)) = d_{min,s}$

Since $\tau \in \left]-\infty, -\frac{\alpha_{bt}\,d_{min,s}}{F_s}\right]$ and $Gain_s(w_s^*) = \tau$.
This leads to $Gain_s(w_s^*) \in \left]-\infty, -\frac{\alpha_{bt}\,d_{min,s}}{F_s}\right]$, thus, $w_s^* \geq L_s\,d_{min,s}$.
Thus, $\forall \tau \in \left]-\infty, -\frac{\alpha_{bt}\,d_{min,s}}{F_s}\right]$, the optimal bandwidth that should be allocated to the swarm without violating the constraint $(Gain_s(w_s^*) \geq \tau)$ is $w_s^* = L_s\,d_{min,s}$.

$$\boxed{\forall \tau \in \left]-\infty, -\frac{\alpha_{bt}\,d_{min,s}}{F}\right], (Gain_s(w_s^*) = \tau) \Rightarrow (w_s^* = L_s\,d_{min,s})}$$

***$lim_2$* and *$Gain_s(lim_2)$*** We have already found that $lim_2 = d_{min,s}$.
Now, we need to calculate $Gain_s(d_{min,s})$ for both Cases II and IV and we should obtain the same value to which we will refer to as: $Gain_s(lim_2)$.

$$Gain_s(d_{min,s}) \overset{\text{case II}}{=} 1 - \frac{d_{min,s}}{L_s\,d_{min,s}} - \frac{\alpha_{bt}\,d_{min,s}}{F_s\,L_s} = 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,d_{min,s}}{F_s\,L_s}$$
$$\overset{\text{case IV}}{=} Gain_s(d_{min,s})$$

We obtain finally that: $Gain_s(lim_2) = 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,d_{min,s}}{F_s\,L_s}$.

**Resolution of the equation** $Gain_s(w_s^*) = \tau, \forall \tau \in \left[-\frac{\alpha_{bt}\,d_{min,s}}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,d_{min,s}}{F_s\,L_s}\right]$

Since $\tau \in \left[-\frac{\alpha_{bt}\,d_{min,s}}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\,d_{min,s}}{F_s\,L_s}\right]$ and $Gain_s(w_s^*) = \tau$.

This leads to $Gain_s(w_s^*) \in \left[-\frac{\alpha_{bt}\, d_{min,s}}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}\right]$, which corresponds to the formula of the gain related to Case II.
Let's try now to invert that formula in order to get an estimation of $w_s^*$:

$$\begin{aligned} Gain_s(w_s^*) = \tau \quad &\overset{\text{case II}}{\Longleftrightarrow} \quad 1 - \frac{w_s}{L_s\, d_{min,s}} - \frac{\alpha_{bt}\, w_s}{F_s\, L_s} = \tau \\ &\Longleftrightarrow \quad 1 - \tau = w_s \left(\frac{1}{L_s\, d_{min,s}} + \frac{\alpha_{bt}}{F_s\, L_s}\right) \\ &\Longleftrightarrow \quad w_s = \frac{(1-\tau) F_s\, L_s\, d_{min,s}}{F_s + d_{min,s}\, \alpha_{bt}} \end{aligned}$$

We can then conclude that:

$$\boxed{\begin{aligned} &\forall \tau \in \left[-\frac{\alpha_{bt}\, d_{min,s}}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}\right], \\ &\qquad\qquad (Gain_s(w_s^*) = \tau) \Rightarrow w_s^* = \frac{(1-\tau)\, F_s\, L_s\, d_{min,s}}{F_s + d_{min,s}\, \alpha_{bt}} \end{aligned}}$$

***$lim_3$* and *$Gain_s(lim_3)$*** the definition of these parameters is very similar to the one done for $lim_4$ and $Gain(lim_4, s)$ in Case A.

Since $w_s$ can only be positive (or equal to 0), we can suppose that $lim_3 = 0$ even thought attaining that limit means that the download might be interrupted.

We need now to calculate $\lim_{w_s^* \to lim_3} Gain_s(w_s^*)$ that will be considered as the upper bound of the gain values:
$\lim_{w_s^* \to lim_3} (Gain_s(w_s^*)) \overset{caseIV}{=} \lim_{w_s^* \to 0} 1 - \frac{1}{L_s} - \frac{\alpha_{bt} w_s^*}{F_s\, L_s} = 1 - \frac{1}{L_s} = Gain_s(lim_3)$

**Resolution of the equation** $Gain_s(w_s^*) = \tau$, $\forall \tau \in \left[1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}, 1 - \frac{1}{L_s}\right[$

We have: $Gain_s(w_s^*) = \tau$ and $\tau \in \left[1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}, 1 - \frac{1}{L_s}\right[$
This means that $Gain_s(w_s^*) \in \left[1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}, 1 - \frac{1}{L_s}\right[$ which corresponds to the formula of the gain related to Case IV.

Let's try now to invert that formula in order to get an estimation of $w_s^*$.

$$\begin{aligned} Gain_s(w_s^*) = \tau \quad & \overset{\text{case IV}}{\Longleftrightarrow} \quad 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, w_s^*}{F_s\, L_s} = \tau \\ & \Longleftrightarrow \quad \frac{\alpha_{bt}\, w_s}{F_s\, L_s} = 1 - \frac{1}{L_s} - \tau \\ & \Longleftrightarrow \quad w_s^* = \frac{F_s\, L_s}{\alpha_{bt}} \left( \frac{L_s\,(1-\tau)-1}{L_s} \right) \\ & \Longleftrightarrow \quad w_s^* = \frac{F_s\,[L_s(1-\tau)-1]}{\alpha_{bt}} \end{aligned}$$

We can then conclude that:

$$\boxed{\begin{aligned} & \forall \tau \in \left[1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}, 1 - \frac{1}{L_s}\right[, \\ & \qquad (Gain_s(w_s^*) = \tau) \Rightarrow w_s^* = \frac{F_s\,(L_s\,(1-\tau)-1)}{\alpha_{bt}} \end{aligned}}$$

**General conclusion for Case B:** The equation: $Gain_s(w_s^*) = \tau$ has the following solution.

$$w_s^* = \begin{cases} L_s\, d_{min,s}, & \forall \tau \in \left]-\infty, -\frac{\alpha_{bt}\, d_{min,s}}{F_s}\right] \\ \frac{(1-\tau) F_s\, L_s . d_{min,s}}{F_s + d_{min,s}\, \alpha_{bt}}, & \forall \tau \in \left[-\frac{\alpha_{bt}\, d_{min,s}}{F_s}, 1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}\right] \\ \frac{F_s\,[L_s(1-\tau)-1]}{\alpha_{bt}}, & \forall \tau \in \left[1 - \frac{1}{L_s} - \frac{\alpha_{bt}\, d_{min,s}}{F_s\, L_s}, 1 - \frac{1}{L_s}\right[ \\ \nexists, & \forall \tau \in \left[1 - \frac{1}{L_s}, +\infty\right[ \end{cases}$$

UNIVERSITAT
ROVIRA i VIRGILI