Reducing Costs in the Personal Cloud: Is BitTorrent a Better Bet?

Rahma Chaabouni, Marc Sánchez-Artigas and Pedro García-López
Universitat Rovira i Virgili, Tarragona (Spain)
{rahma.chaabouni|marc.sanchez|pedro.garcia}@urv.cat

***Abstract*—Lately, Personal Cloud storage services, like Dropbox, have emerged as user-centric solutions that provide easy management of the users' data. To meet the requirements of their clients, such services require a huge amount of storage and bandwidth. In an attempt to reduce these costs, we focus on maximizing the benefit that can be driven from the interest of users in the same content by the introduction of the BitTorrent protocol. In general, it is assumed that BitTorrent is only effective for large files and/or large swarms, while the client-server approach is more suited for small files and/or small swarms. However, there is no concrete study on the comparative efficiency of both protocols for small files yet.**

In this paper, we study the download time and offload ratio in BitTorrent compared to HTTP. Based on this study, we propose an algorithm for the management of these protocols. The choice of the protocol is made based on the prediction of the efficiency of BitTorrent and HTTP for each case. We validate our algorithm on a real trace of the Ubuntu One file service, achieving important savings in the cloud bandwidth without degrading the download time.

***Keywords*-Personal cloud, bittorrent, content distribution**

I. Introduction

Cloud storage services have become very popular these days as a paradigm that enables individuals and organizations to store, edit and retrieve data stored in remote servers and which can be accessed all over the Internet. Such systems are generally equipped with a set of features that allow sharing and collaboration between the users. That is why, nowadays, millions of users are putting their files into online cloud storage systems (like Dropbox, Google Drive or Box ...) in order to be able to access them wherever they go or as a backup.

When a client adds a new file to his personal folder, the content is divided into small entities called "chunks" that are pushed to the storage servers while the meta-data is kept in a different server. Once the file is uploaded, all the synchronized devices will be notified of the addition of the new file and will request a copy of it from the cloud storage servers. This will result in the repeated distribution of the same content in a short period of time. To avoid that, the cloud can benefit from the clients upload capacities to save its bandwidth.

In this context, we propose to introduce the BitTorrent [1] protocol when the load on a specific file becomes high. In fact, the efficiency of the peer-assisted paradigm 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 cloud from doing all the serving.

Unfortunately, the use of BitTorrent may incur a longer download time compared to HTTP especially for small files. According to previous studies, these small files form a very high percentage of the data stored in personal cloud servers; 99% of the files are of size smaller than 16 MB according to [2]. The main challenge is to decide when it is worth switching to BitTorrent. The key elements in making the decision are the gain in download time and the peers' contribution. The former represents the difference in download time between HTTP and BitTorrent. The latter measures the total amount of data that can be obtained from the peers. To our knowledge, there were no previous studies that compare the BitTorrent and HTTP protocols for distributing small files. Thus, it comes the need to draw a complete comparative study between both protocols.

In this paper, we first study the download time in HTTP and BitTorrent and measure the efficiency of each protocol. Then, we propose a dynamic switching algorithm that can be applied in real personal cloud systems.
Our key contributions are the following:

- 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 analytical 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 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 validate all the proposed formulas with focus on small files.
- We 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.
- We analyze a public trace of the Ubuntu One system and study the users and files characteristics and the access patterns. Later, we apply our algorithm on the trace and measure the amount of data that can be offloaded based on different time constraints. We notice that the overall

offloaded data volume exceeds 16% of the total amount of data exchanged. From an economic point of view, this corresponds to savings of the order of hundreds to thousands of dollars per month. We also study the effect of file bundling on the trace and notice that it can only improve the overall offload by a small increment.

The remainder of this paper is organized as follows: we discuss related work in Section II and we state the problem in Section III. In Section IV, we propose an estimation of the distribution time in BitTorrent and introduce the metrics needed to evaluate the efficacy of HTTP and BitTorrent. Section V presents our proposed algorithm for the management of the download protocols. In Section VI, we evaluate the accuracy of the proposed formulas and the efficacy of the algorithm based on a real trace of the Ubuntu One system. Finally, section VII concludes the paper and presents our future plans.

II. Related Work

Integrating BitTorrent with the cloud is not a new idea in the literature. Several previous studies have tried to combine BitTorrent content distribution technologies with Cloud environments. For instance, in [3], the authors 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 may reduce costs when transferring large amounts of data. Their experiments with an Amazon S3 seed and several external seeds show that S3 contributes a large percentage of the data volume (around 50% with three external seeds). They also cancel the S3 seed and show that with their external seeds, they only incur in an 8% increase in download time while eliminating cloud transfer costs.

Many previous works have focused on reducing download times for large contents using BitTorrent in Cloud settings. BitTorrent has proven its efficiency not only for bulk synchronous content distribution [4] but also for reducing transfer times for cloud virtual images [5], [6], [7]. For example, in [7], authors demonstrate that their BitTorrent-based solution for distributing virtual machines delivers up to an 30x speedup over traditional remote file system approaches.

In our previous works [8] and [9], we presented some techniques that can be used to apply the approach particularly in cloud storage systems. In [8], we propose to transparently switch from HTTP to BitTorrent upon detection of a certain critical mass demand on a specific content. The threshold is placed on the number of users requesting the same files. The system tests with each new request whether the current number of requesters of the corresponding file passes the predefined threshold or not. When the threshold is reached, the system decides to adopt BitTorrent instead of HTTP in order to avoid bottlenecks from the one hand, and to save the cloud bandwidth from the other hand. In [9], we focus more on the allocation of the data-center bandwidth. This paper is a continuation of this previous work with more analysis and experimental evaluation of the threshold at which the system should switch from HTTP to BitTorrent.

For such analysis, it is essential to study both protocols and draw a full comparison of client-server versus peer assisted file distribution. In this context, Wei et al. [10] 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 come to the conclusion that BitTorrent outperforms FTP when the file size is greater than 20 MB. However, this limit is only relative to their specific deployment setup and cannot be generalized in other settings such as regular clients in a wide area network. Another prominent work is the estimation of the minimum distribution time of Kumar and Ross in [11]. Such estimation is useful to get a prediction of the time needed to distribute a certain content from a set of seeds to a set of leechers.

Finally, there are few studies related to the data characteristics and usage patterns in personal clouds systems. Drago et al. present in [12] a characterization of Dropbox including typical usage, traffic patterns, and possible performance bottlenecks. Another example is [2] where the authors study the characteristics of the data stored in a personal cloud system for campus students. They notice that 99% of the files are smaller than 16 MB, and that most of I/O requests are for small files.

III. Problem Description

We consider a classic personal cloud system where the storage server is responsible for storing the clients files and managing the corresponding requests. Each client i has an upload bandwidth u_i and a download bandwidth d_i.

The two following common file distribution scenarios could benefit from our hybrid download strategy:

1. *Synchronization:* User A is adding a new file f to his personal account. During the synchronization process, the same file will be download by all the other synchronized devices of the user. (Part (a) of Fig. 1)
2. *Sharing:* User A is sharing a file f with other users. In this case, the file will be downloaded by all the synchronized devices of the users. (Part (b) of Fig. 1)

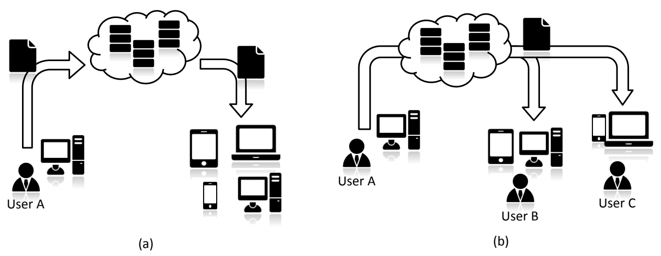


Fig. 1: Synchronisation and sharing in personal cloud systems

Both cases can be modeled by the problem of distributing a file f of size F from the cloud server to L distinct nodes. We denote by $\mathcal{S}$ the set of cloud seeds serving f and by $u(\mathcal{S})$ their aggregated upload speed dedicated to f. In personal clouds, file synchronization and content distribution follow a client-server model centralized in the cloud storage provider. The download protocol adopted by these providers is HTTP. The

TABLE I: Measured download times for small files using HTTP and BitTorrent. The seed bandwidth is limited to 5 Mbps and the clients are homogeneous each having an upload and download speed of respectively 1 and 2 Mbps.

Clients count	1 MB file				5 MB file				10 MB file			
	HTTP	BT	Time difference	Data from peers	HTTP	BT	Time difference	Data from peers	HTTP	BT	Time difference	Data from peers
2	4 s	5.51 s	-1.51 s	236.13 KB	20 s	21.52 s	-1.52 s	2.9 MB	40 s	42.06 s	-2.06 s	6.08 MB
3	4.8 s	5.47 s	-0.67 s	819.6 KB	24 s	21.69 s	+2.31 s	6.02 MB	48 s	42.73 s	+5.27 s	11.97 MB
4	6.4 s	6.03 s	+0.37 s	1.57 MB	32 s	23.06 s	+8.94 s	7.84 MB	64 s	42.83 s	+21.17 s	17.6 MB
5	8 s	6.25 s	+1.75 s	1.64 MB	40 s	24.05 s	+15.95 s	11.59 MB	80 s	44.68 s	+35.32 s	23.64 MB

choice of this protocol is made because it uses the port 80 which is generally kept open. While this kind of architecture (client-server) is appropriate for some uses cases, it is not optimal when the number of nodes requesting the same content is high, as it might result in bandwidth bottlenecks in the cloud.

A possible solution is to benefit from the high number of requests and use the clients' spare upload bandwidth to offload the server. The main idea is to switch from HTTP to BitTorrent upon detection of an increasing number of requests on a specific content. While this switch seems to be very convenient for big files, it might incur a significant increase in download time for the small ones.

The main challenge is to identify the best switching point that will help avoiding bottlenecks without affecting significantly the download time. There are many important parameters that should be considered when choosing this point, including: the size of the shared file, the bandwidth of the cloud allocated to that file, the number of peers downloading the file and their corresponding bandwidth capacities. To this extent, the choice of the switching point should be based on a complete comparative study of BitTorrent and HTTP in order to determine the most convenient one in each specific case. This study should be able to answer the following question: *How much time would the clients gain (or lose) and how much bandwidth could the cloud save, if the download protocol is switched from HTTP to BitTorrent?*

For the rest of the paper, we will consider the following notation:

- F: size of the requested file f
- $\mathcal{S}$: set of providers of f (seeder nodes)
- $\mathcal{L}$: set of requesters of f (leecher nodes), L= $|\mathcal{L}|$ is the number of requesters
- $\mathcal{I}$: set of all the nodes, $\mathcal{I} = \mathcal{L} \cup \mathcal{S}$
- u_i: upload speed of node $i \in \mathcal{L}$
- d_i: download speed of node $i \in \mathcal{L}$
- $d_{min} = \min\limits_{i \in \mathcal{L}}(d_i)$: download speed of the slowest leecher requesting f
- $u(\mathcal{A}) = \sum\limits_{i \in \mathcal{A}} u_i$: aggregated upload bandwidth of $\mathcal{A} \subseteq \mathcal{I}$
- $d(\mathcal{A}) = \sum\limits_{i \in \mathcal{A}} u_i$: aggregated download bandwidth of $\mathcal{A} \subseteq \mathcal{I}$
- $C_f = \{(u_i, d_i),\ \forall i \in \mathcal{L}\}$: set of upload and download bandwidths of all the leechers interested in f.

IV. Client-server Versus Peer-Assisted File Distribution

It is commonly believed that BitTorrent is not convenient for the distribution of small files. But, to our knowledge, there is no proof of such assumption. Wei et al. noticed in [10] that, in their specific experimental settings, BitTorrent outperforms the FTP protocol only when the file size is greater than 20 MB. However, in practice, we found that BitTorrent can be efficient for small files. We ran several experiments distributing files of sizes 1, 5 and 10 MB using a unique seed. We used the following common ADSL bandwidth settings: the clients had an upload bandwidth u_i=1 Mbps, a download bandwidth d_i=2 Mbps and the bandwidth allocated by the cloud to each exchanged file was $u(\mathcal{S})$=5 Mbps. We measured the average download time in HTTP and BitTorrent for each experiment and calculated the corresponding gain or loss in download time. We also measured the total amount of data contributed by the peers. We report the results in table I. All the download times in the table are in seconds. We notice that, with four clients downloading a 1 MB file, BitTorrent can reduce the download time compared to HTTP. The peers contribution can reach 40% of the total data volume in some cases.

In this section, we present our estimation for the distribution time of small files via BitTorrent. We also introduce the gain and offload ratios in order to measure the trade-off between this protocol and HTTP.

A. The Distribution Time for Small Files in BitTorrent

1) Background: To get an estimation of the download time in BitTorrent-like systems, we borrow the following formula proposed in [11] by Kumar and Ross:

$$T_{pa}\left(u(\mathcal{S}), C_f, F\right) = \frac{F}{\min\left\{d_{min}, \frac{u(\mathcal{I})}{\mathrm{L}}, u(\mathcal{S})\right\}}, \qquad (1)$$

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

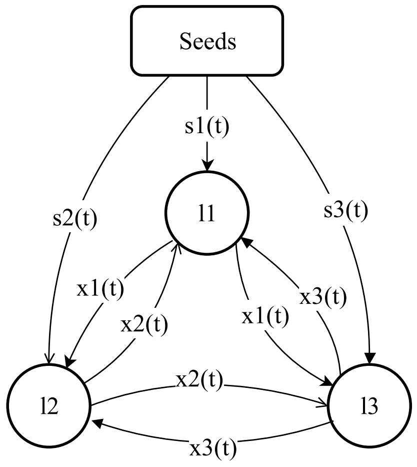


Fig. 2: General distribution scheme structure: Leecher li ($i \in \{1, 2, 3\}$) downloads "fresh" data at the rate $s_i(t)$ from the seeds. The data is replicated later to the other 2 leechers at a rate $x_i(t) < s_i(t)$.

1) Case A: $d_{min} \leq \min\left\{\frac{u(\mathcal{I})}{\mathrm{L}}, u(\mathcal{S})\right\}$ and $d_{min} \leq \frac{u(\mathcal{L})}{\mathrm{L}-1}$: In this case, the download speed of the peers is limited by the download bandwidth of the slowest peer in the swarm d_{min}.
2) Case B: $d_{min} \leq \min\left\{\frac{u(\mathcal{I})}{\mathrm{L}}, u(\mathcal{S})\right\}$ and $\frac{u(\mathcal{L})}{\mathrm{L}-1} \leq d_{min}$: In Case B, the transfer is limited by the maximum speed at which a leecher can get data from the other leechers, that is $\frac{u(\mathcal{L})}{\mathrm{L}-1}$.
3) Case C: $\frac{u(\mathcal{I})}{\mathrm{L}} \leq \min\{d_{min}, u(\mathcal{S})\}$: The transfer bottleneck in this case is limited by the aggregated upload speed of the network $u(\mathcal{I})$ divided equally between the L leechers.
4) Case D: $u(\mathcal{S}) \leq \min\left\{d_{min}, \frac{u(\mathcal{I})}{\mathrm{L}}\right\}$: In this case, the upload bandwidth of the seed $u(\mathcal{S})$ is the maximum limit at which each peer can download "fresh" content.

For each of the cases listed above, the authors in [11] constructed a seeding rate profile $s_i(t)$ which denotes the bit rate at which the seeds send pieces to leecher i at time t. The adopted distribution scheme is the following: As soon as a leecher li begins to receive data from the seed, it replicates it to each of the other $(\mathrm{L}-1)$ leechers at a rate $x_i(t)$, where $x_i(t) \leq s_i(t)$, as shown in Figure 2. For each case, the distribution scheme consists of L application-level multicast trees, each rooted at a specific seed, passing through one of the leechers and terminating at each of the L-1 other leechers.

To calculate the offload ratio in the following section, we need to measure the volume of data offloaded from the cloud. We present here the seeding rate for each case. This rate, denoted by $s_i(t)$ for the sake of clarity, depends on the time t, the file size F, the upload speed of the seeds $u(\mathcal{S})$, and the set of upload and download speeds of all the leechers C_f. For a complete proof and more details regarding these formulas, we kindly refer the reader to the original paper [11].

TABLE II: Estimated versus experimental distribution time with BitTorrent of a 1MB file.

Clients count	2	3	4	5
Estimated time	4 s	4 s	4 s	4 s
Experimental time	5.51 s	5.47 s	6.03 s	6.25 s
Absolute error	1.51 s	1.47 s	2.03 s	2.25 s
Relative error	37.75%	36.75%	50.75%	56.25%

$$s_i(t) = \begin{cases} \frac{u_i \times d_{min}}{u(\mathcal{L})} & \underline{\textit{Case A}} \\ \frac{u_i - u(\mathcal{L})}{\mathrm{L}-1} + d_{min} & \underline{\textit{Case B}} \\ \frac{u_i - u(\mathcal{L})}{\mathrm{L}-1} + \frac{u(\mathcal{I})}{\mathrm{L}} & \underline{\textit{Case C}} \\ \frac{u_i \times u(\mathcal{S})}{u(\mathcal{L})} & \underline{\textit{Case D}} \end{cases} \quad (2)$$

2) Adding the BitTorrent overheads: One of the limitations of (1) is that it does not take into consideration the overhead that peer-assisted systems may present compared to the client-server ones. These overheads may be neglected for large files. 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 ran several experiments distributing a 1MB file to several clients. We considered swarms whose size ranged from 2 to 5. We considered the same bandwidth settings as in the experiments used in Table I. We measured the experimental download times and compared them to the estimated ones using (1). We calculated also the absolute and relative errors. We group all these results in Table II, where the estimated and experimental download times, and the absolute error are all measured in seconds.

As we can see in Table II, the difference between the estimated and experimental results can exceed 50% in some cases, which proves that an accurate estimation should include the protocol overheads. These overheads can be mainly of two types, each related to a different phase of BitTorrent:

- *Overhead related to the start-up phase:* Before starting the download, 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. After locating and connecting to the peers, the leecher can finally begin the transfer.
  This overhead is relative to the architecture of the system. It can be monitored and dynamically adapted based on the load of the system. We experimentally studied this overhead and noticed that it can be simply modeled as a constant duration α_{bt} added to the download time. For more details about the experimental evaluation of α_{bt}, please refer to Fig. 3.
- *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 in [13], where the authors introduced a parameter to scale down the upload speed of leechers. This parameter, denoted as $\eta \in [0, 1]$, measures the effectiveness of file sharing. It can be computed as follows:

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

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

$$\eta = 1 - \sum_{n_i=0}^{N-1} \frac{1}{N} \left(\frac{N - n_i}{N\,(n_i + 1)} \right)^{k},$$

where N is the number of pieces of the served file and k the number connections a peer has.

The authors in [13] focused on the case of large files and concluded that $\eta \approx 1$ when N is high. Let us now consider a small file of 1MB composed of $k = 4$ chunks each of 256KB. For $N = 2$, the above equation yields $\eta = 0.7069$, 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.

Considering the above listed overheads, we were able to extend Eq. (1) in order to provide an accurate estimation of the download time in BitTorrent as follows:

$$T_{bt}\left(u(\mathcal{S}), C_f, F\right) = \frac{F}{\min\left\{d_{min}, \frac{u^{'}(\mathcal{I})}{\mathrm{L}}, u(\mathcal{S})\right\}} + \alpha_{bt}, \quad (3)$$

where $u^{'}(\mathcal{I}) = u(\mathcal{S}) + \eta\ u(\mathcal{L})$ is the scaled aggregated upload speed of all the nodes, including both the seeders and the leechers.

B. Comparative Analysis of BitTorrent Relative to HTTP

We introduce here two metrics that measure the comparative efficiency of HTTP and BitTorrent, especially for small files. These metrics are the gain and offload ratios. The former represents the normalized ratio of the difference between download times. The latter models the amount of data offloaded from the cloud.

1) Gain Ratio: To measure the difference between the download times of client-server and peer-assisted systems, we introduce the gain ratio as follows:

$$\mathit{Gain}(u(\mathcal{S}), C_f, F) = \frac{T_{cs}(u(\mathcal{S}), C_f, F) - T_{bt}(u(\mathcal{S}), C_f, F)}{T_{cs}(u(\mathcal{S}), C_f, F)},$$

where T_{cs} is the distribution time in a client-server architecture. T_{cs} is limited by the download speed of the slowest peer d_{min} or the bandwidth of all the seeds $u(\mathcal{S})$ divided equally between the L clients. T_{cs} can be simply defined as follows:

$$T_{cs}\left(u(\mathcal{S}), C_f, F\right) = \frac{F}{\min\left\{d_{min}, \frac{u(\mathcal{S})}{\mathrm{L}}\right\}}. \quad (4)$$

Clearly, the gain can take negative or positive values and can be also equal to zero. For instance, if the gain is positive, this means that downloading the file via BitTorrent takes less time than using HTTP. To derive the equation of the gain, we distinguish four different cases based on the values of $\min\left\{d_{min}, \frac{u'(\mathcal{I})}{\mathrm{L}}, u(\mathcal{S})\right\}$ and $\min\left\{d_{min}, \frac{u(\mathcal{S})}{\mathrm{L}}\right\}$:

1) Case I: $d_{min} \leq \frac{u(\mathcal{S})}{\mathrm{L}}$ and $d_{min} \leq \min\left\{\frac{u'(\mathcal{I})}{\mathrm{L}}, u(\mathcal{S})\right\}$:
In this case, the bottleneck in HTTP and BitTorrent is the download speed of the slowest peer. The corresponding download times are: $T_{cs} = \frac{F}{d_{min}}$ and $T_{bt} = \frac{F}{d_{min}} + \alpha_{bt}$.
2) Case II: $\frac{u(\mathcal{S})}{\mathrm{L}} \leq d_{min}$ and $d_{min} \leq \min\left\{\frac{u'(\mathcal{I})}{\mathrm{L}}, u(\mathcal{S})\right\}$:
In Case II, the bottleneck in HTTP is $\frac{u(\mathcal{S})}{\mathrm{L}}$, while it is equal to d_{min} in BitTorrent. The corresponding download times are: $T_{cs} = \frac{F \times \mathrm{L}}{u(\mathcal{S})}$ and $T_{bt} = \frac{F}{d_{min}} + \alpha_{bt}$.
3) Case III: $\frac{u'(\mathcal{I})}{\mathrm{L}} \leq \min\{d_{min}, u(\mathcal{S})\}$:
In this case, the bottleneck in BitTorrent is $\frac{u'(\mathcal{I})}{\mathrm{L}}$. And since $u(\mathcal{S}) \leq u^{'}(\mathcal{I})$ and $\frac{u'(\mathcal{I})}{\mathrm{L}} \leq d_{min}$, this means that $\frac{u(\mathcal{S})}{\mathrm{L}}$ is always $\leq d_{min}$. Thus, in this case, $T_{cs} = \frac{F \times \mathrm{L}}{u(\mathcal{S})}$ and $T_{bt} = \frac{F \times \mathrm{L}}{u'(\mathcal{I})} + \alpha_{bt}$.
4) Case IV: $u(\mathcal{S}) \leq \min\left\{d_{min}, \frac{u'(\mathcal{I})}{\mathrm{L}}\right\}$:
Since $\frac{u(\mathcal{S})}{\mathrm{L}} \leq u(\mathcal{S})$ and $u(\mathcal{S}) \leq d_{min}$, this means that $\frac{u(\mathcal{S})}{\mathrm{L}}$ is always $\leq d_{min}$. In this case, $T_{cs} = \frac{F \times \mathrm{L}}{u(\mathcal{S})}$ and $T_{bt} = \frac{F}{u(\mathcal{S})} + \alpha_{bt}$.

For each of the previous cases, we substitute T_{cs} and T_{bt} to derive the gain ratio as follows:

$$\mathit{Gain}\left(u(\mathcal{S}), C_f, F\right) = \begin{cases} -\frac{\alpha_{bt} \times d_{min}}{F} & \underline{\textit{Case I}} \\ 1 - \frac{u(\mathcal{S})}{\mathrm{L}.d_{min}} - \frac{\alpha_{bt}.u(\mathcal{S})}{F \times \mathrm{L}} & \underline{\textit{Case II}} \\ 1 - \frac{u(\mathcal{S})}{u'(\mathcal{I})} - \frac{\alpha_{bt}.u(\mathcal{S})}{F \times \mathrm{L}} & \underline{\textit{Case III}} \\ 1 - \frac{1}{L} - \frac{\alpha_{bt} \times u(\mathcal{S})}{F \times \mathrm{L}} & \underline{\textit{Case IV}} \end{cases} \quad (5)$$

2) Offload Ratio: The offload ratio defines the amount of data offloaded from the cloud seed. It is determined by the total amount of data exchanged between the peers divided by the total downloaded data volume:

$$\begin{aligned} \mathit{Offload}\left(u(\mathcal{S}), C_f, F\right) &= \frac{\textit{data from peers}}{\textit{total data sent}} \\ &= 1 - \frac{\textit{data from cloud}}{\textit{total data sent}} \\ &= 1 - \frac{\sum_{i \in \mathcal{L}} \int_0^{T_{bt}(u(\mathcal{S}), C_f, F)} s_i(t)dt}{F \times \mathrm{L}}, \end{aligned}$$

where $s_i(t)$ is the seeding rate. Taking into consideration the seeding rate as defined in (2), we can deduce the offload rates as follows:

[1]The rectified version of [13] which contains the correct expression of η can be found at: http://users.encs.concordia.ca/~dongyu/paper/bittorrent.pdf

$$\textit{Offload}\,(u(\mathcal{S}), C_f, F) = \begin{cases} 1 - \frac{1}{\mathrm{L}} & \underline{\textit{Case A}} \\ \frac{\eta . u(\mathcal{L})}{\mathrm{L} \times d_{min}} & \underline{\textit{Case B}} \\ 1 - \frac{u(\mathcal{S})}{u'(\mathcal{I})} & \underline{\textit{Case C}} \\ 1 - \frac{1}{\mathrm{L}} & \underline{\textit{Case D}} \end{cases} \qquad (6)$$

V. Switching Algorithm

In this section, we first study the criteria that can be considered in the definition of the switching point. We present later our proposed algorithm for the management of the download protocols.

A. Switching Criteria

We presented in the previous section two key parameters that can help us measure the tradeoff between HTTP and BitTorrent. The gain ratio measures the gain or loss in time that the leechers might experience when switching from HTTP to BitTorrent. The offload ratio gives an estimation of the amount of data that can be offloaded from the server thanks to BitTorrent.

It is clear that if we neglect a potential increase in download time caused by the switch to BitTorrent, the overall offload ratio will always be the highest possible. However, it is equally important to not degrade significantly the download service for the clients. We distinguish the four following cases based on the constraints that can be placed on these parameters:

i. The first possible solution is to put no constraints, that is, BitTorrent is always used when the number of leechers L$\geq$ 2. In this case, the overall offload ratio will be the highest possible. But, the clients might experience a longer download time.
ii. Another possible solution is to put a limit on the offload ratio: the cloud switches to BitTorrent only when the offload is important. For example, the cloud can decide to switch only when the estimated offloaded bandwidth is above 50% of the total bandwidth, regardless of the download time.
iii. The third possible case is fixing a gain limit: the cloud 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.
iv. The last possibility is fixing both the gain and offload ratios. While this case presents an efficient strategy to avoid unnecessary switches, it might be too strict and could limit the overall offload ratio.

After listing all the possible scenarios, we believe that the most convenient procedure to manage the download protocols is the third one. To this extent, we pose τ as the gain constraint. 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 τ 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.

τ can take different values depending on the type of the user account. The choice of its concrete value is left up to the system administrator depending on his needs. A possible concrete example of τ 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 those who are paying money for the service. For this type of clients, τ should be always ≥ 0. However, for free users, which represent a significant portion of the overall user mass[2], it is possible to loosen that constraint, and tolerate delays of up to 50% (which corresponds to $\tau = -0.5$), for instance.

To get an idea about the tradeoff between HTTP and BitTorrent, please refer to Table III in which we measured the overall offload ratio based on different values of τ.

B. Implementation of the Switching Algorithm

For the management of the download protocols, we propose Algorithm 1, which is executed upon the arrival of each new download request on a certain file.

Algorithm 1 Protocol Decision Algorithm

```
Require: $\tau$: the gain constraint
Require: $switched_f$: the state of file $f$
Require: $F$: the size of file $f$
Require: $u(\mathcal{S})$: the upload speed of the seeder nodes
Require: $C_f = \{(u_i, d_i),\ \forall i \in \mathcal{L}\}$: set of upload and
  download bandwidths of all the leechers interested in $f$.
  if ( not $switched_f$ ) then
    calculate $Gain(u(\mathcal{S}), C_f, F)$
    if ($Gain(u(\mathcal{S}), C_f, F) \geq \tau$ ) then
      create a .torrent
      launch a BT seed in the cloud
      for all clients requesting $f$ do
        get the .torrent from the server
        launch a BT leecher
        start BT transfer
      end for
      $switched_f$=true
    else
      download the file via HTTP
    end if
  else
    send the .torrent to the new requester
    launch a BT leecher inside that requester
  end if
```

We suppose that our system keeps track of the state of each file f as a boolean value *switched*$_f$, where *switched*$_f$=*true* if

[2] 96% of Dropbox clients use the free version of the service (Souce: http://www.economist.com/blogs/babbage/2012/12/dropbox)

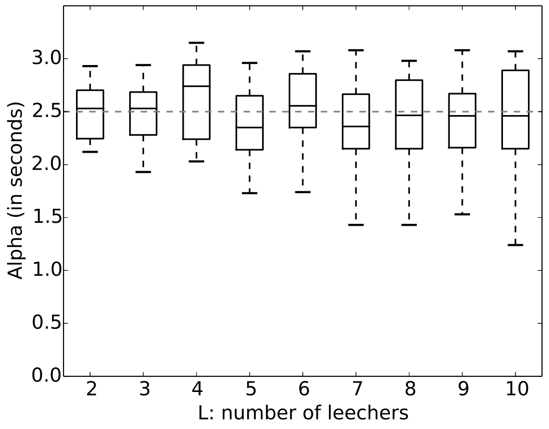


Fig. 3: The overhead α_{bt}: time before clients start downloading for a file of 1 MB size

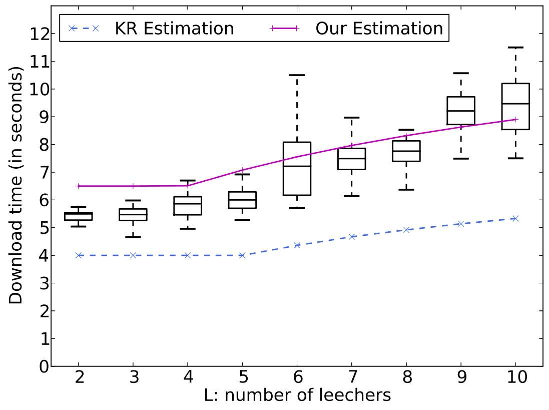


Fig. 4: Comparison of the experimental download time (box-plot), our estimation and the estimation proposed in [11] (KR Estimation)

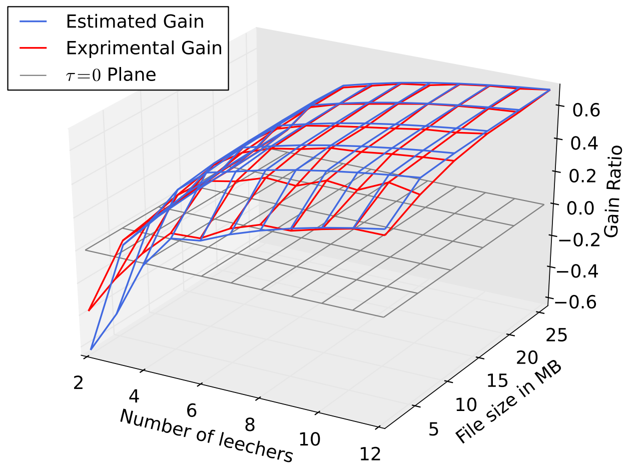


Fig. 5: Experimental versus estimated gain ratios and the $\tau = 0$ gain plane

the current download protocol is BitTorrent (the switch has already taken place) and *false* otherwise.

The algorithm works as follows: Whenever there is a new download request on a file f, the system verifies the download protocol already in use to distribute f. If it is being downloaded by the default protocol (HTTP), the system computes the estimated gain and compares it with τ. If the resulting gain is below the constraint, the file will be sent to the requester via HTTP. Otherwise, the distribution protocol will be switched to BitTorrent. A *.torrent* file will be created and sent to all the clients requesting f. In parallel, a seed will be launched in the cloud. Upon the reception of the *.torrent* file, a BitTorrent leecher will be launched inside each of these clients. After this phase, the clients will start downloading the file in BitTorrent, while offloading the cloud from doing all the serving.

VI. Validation

In this section, we present our experimental results. In the first part of this section, we verify the accuracy of the previously proposed formulas: the distribution time in BitTorrent and the gain and offload ratios. The second part is dedicated to the validation of the algorithm: we study a trace of the Ubuntu One (UB1) system and measure the amount of bandwidth that can be saved if we apply the algorithm.

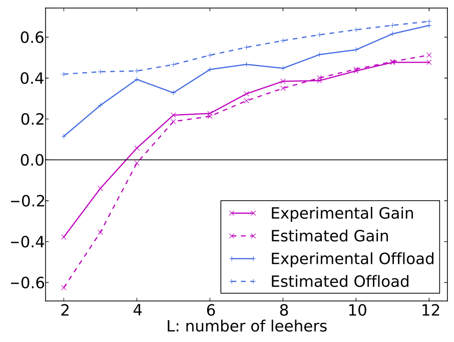
A. Part I: Validation of the formulas

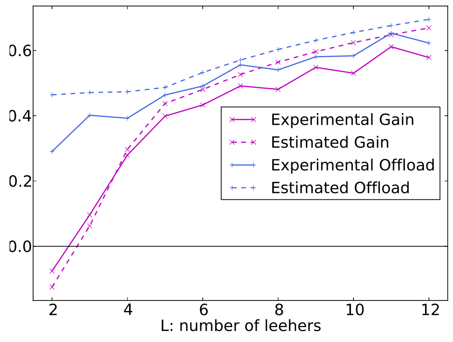
1) The Overhead α_{bt} and the Download Time in BitTorrent: To validate our extended formulas of the download time, we run repeated experiments using a 1 MB file. The experimental scenario is to distribute the file via BitTorrent starting with a unique seed. The reasons behind the choice of such a small file lies in the fact that in personal cloud systems most of the files are in the order of a few megabytes in size.
The experimental setting is the following: the upload bandwidth of the cloud dedicated for the file is $u(\mathcal{S}) = 5$ Mbps. The number of clients ranges from 2 to 10. Each of them has an upload bandwidth of 1 Mbps and a download bandwidth of 2 Mbps.

We repeated each experiment 5 times and measured the average values of the download time and the overhead α_{bt} for each client. Figure 3 represents a box-plot of the time interval between the moments when the clients are launched and when they start downloading the file. This time interval corresponds to α_{bt}. We notice that the average value of the overhead is about 2.5 seconds for our architecture. Using this value for the discovery overhead ($\alpha_{bt} = 2.5$), Figure 4 compares our estimation, the one proposed in [11] and the experimental results. We can clearly see that the error can reach 40% in the case of Kumar and Ross's estimation. This error is reduced to about 10% using the estimation we propose.

2) Gain and Offload Ratios: Since the gain is a key parameter in the protocol decision algorithm, it is important to verify the accuracy of our estimation compared to real experimental values.

We ran experiments using the same bandwidth distribution


(a) 1 megabyte file


(b) 5 megabyte file

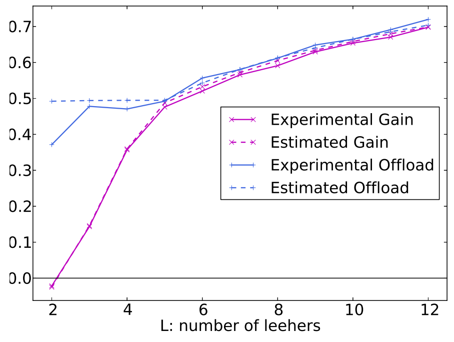
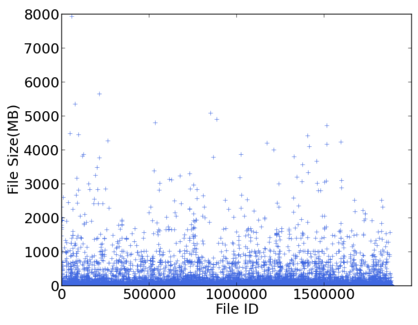
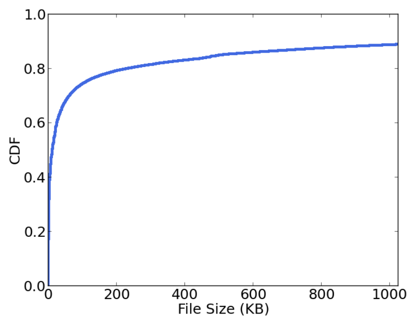

(c) 25 megabyte file

Fig. 6: Estimated versus experimental gain and offload ratios for files with different sizes


(a) File sizes distribution


(b) CDF file sizes

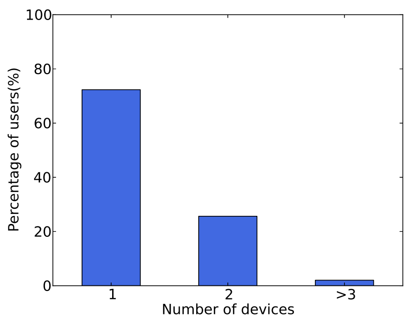

(c) Number of devices per account

Fig. 7: Data and User characteristics

as mentioned above. The goal is to compare the experimental and the estimated gain ratios when distributing a file to a set of nodes whose size range from 2 to 12 nodes. The file size varies from 1 to 25 MB. Figure 5 represents a 3-dimensional plot of the results. The *Gain*= 0 plane represents the threshold $\tau = 0$. As we can see , the experimental and estimated surfaces are very close and the difference between them is slight.

Figures 6a, 6b and 6c present some vertical sections of the previous plot for files of size 1, 5 and 25 megabytes along with the corresponding estimation and experimental values of the offload ratio. We notice that the estimations are very close to the experimental results in most cases. For instance, for the smallest file of size 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 negligible for swarms of size ≥ 4 clients. For bigger files, the estimation is very accurate and the error does not exceed 5% in most cases.

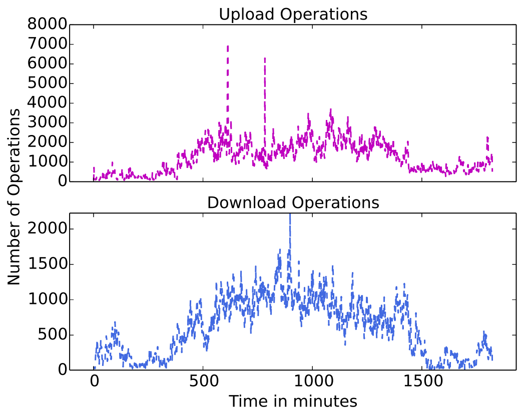
B. Part II: Validation of the algorithm

1) Ubuntu One trace: To validate our proposal and measure how much bandwidth can be saved using our algorithm, we used a real trace [3] of UB1 [14]. The trace was collected based on the behavior of real users, each represented by a hash code for privacy reasons. Each line of the trace represents an operation of download or upload performed by a user on a file. For each operation, several information were collected, including: the time-stamp, the type of operation ('up' or 'down'), the hash and size of the file in question and finally the identifier of the user. For a period of 30 hours, 3,318,950 operations on 1,887,247 distinct files were logged including 2,231,791 upload operations (67.24%) and 1,087,159 (32.76%) download operations. The total downloaded volume was about 1,240.25 GB. The total number of different users involved in this trace was 19,319.

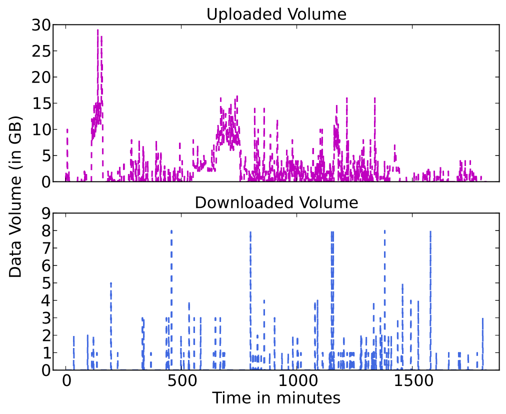
a) Data and User Characteristics: In this section, we study the file distribution based on the data collected in the trace. Figure 7a represents the sizes of all the files included in the trace. These sizes range from 0 bytes (for folders) to over 7 GB. Figure 7b represents the portion of the CDF of the file sizes smaller than 1 MB. We notice that about 90% of the files are less than 1 MB in size. To be more concrete, we find out that 92.88% of the files are smaller than 1 MB, 4.53% are between 1 and 5 MB and only about 2.59% of files are over 5 MB. Figure 7c presents the distribution of the number of devices per client. We notice that 72.32% of the clients have only one device, 25.64% have 2 devices and only 2.04% have more than 3 devices.

It is important to signal here that the trace is not very

[3] Inquires about the trace should be sent via Email to the first author.



(a) Upload vs. download operations



(b) Uploaded vs. downloaded data volume

Fig. 8: Input/Output traffic over 30 hours

favorable to us since more that 70% of the clients use the UB1 service to upload content only, thereby limiting the benefits of our switching mechanism to a reduced number of cases.

b) Input/Output Patterns: To characterize the UB1 workload, we plot in Figure 8a the total number of upload and download operations per minute. Also, the total amount of data uploaded and downloaded during the same time interval is plotted in Figure 8b.

We notice that the upload operations are more frequent than downloads and the gap between the two curves is high especially in terms of data volume. This indicates that there are few accesses to the files after being uploaded, which leads to the conclusion that most of the users are using this service only for backup.

2) Results: We applied our algorithm on the trace using the following settings:

- The upload speed of the seed: $u(\mathcal{S}) = 2$ Mbps. We remind our reader that $u(\mathcal{S})$ 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.
- The clients are homogeneous and have an upload and download speed of 512 Kbps and 1 Mbps, respectively.
- 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. Our goal was 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: $t_2 - t_1 \leq T_{cs}$. T_{cs} is calculated based on the settings listed above. After the identification of these files, we calculated for each case the gain ratio using (5). Depending on the gain value and the τ constraint, we identified the files that were subject to switching and measured the corresponding offloaded volume of data using (6).

Table III 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 (1,240.25 GB). We varied the values of the switching constraint τ in order to get a global idea of the gains, and we noticed that if we fixed τ to tolerate losses of 20% ($\tau = -0.2$), the cloud load could be reduced up to 16%. In the case of stricter constraints, e.g., no loss is tolerated ($\tau = 0$), or no switch unless we gain 20% in download time ($\tau = 0.2$), the overall offload percentage falls down to around 11%.

TABLE III: Offloaded volume and offload percentage resulting from the application of Algorithm 1 using different τ 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%

Even though the UB1 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 have more users and more file sharing.

Monetary Cost: To measure the amount a money that can be saved using our algorithm, we consider a cloud storage system that uses Amazon Simple Storage Service (S3) as a storage back-end.

At the time of writing this paper, the standard charging rates for data transfer were[4]:

- \$0.0 per GB for the first 1 GB/month
- \$0.12 per GB for transfers up to 10 TB/month
- \$0.09 per GB for the next 40 TB/month

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

TABLE IV: Results using file bundling

Bundling 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 %

Using these rates, the overall data transfer cost is approximately $\$3,000$ per month. Fixing the gain constraint to $\tau = -1$ would lead to savings of about $\$450$ per month which is about $\$5,374$ per year. These savings will be higher for systems that involve more sharing than UB1.

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 [15] in an attempt to reduce both transmission latency and control overhead.

If we take a look at equation (5), we notice that the gain ratio and the file size F are related in a way that if F increases, the gain 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 IV shows the results of grouping files considering 2 different bundling periods. We notice that, compared to the previous results, 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.

VII. Conclusion

In this paper, 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 cloud systems. 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 benefit that can be driven from the switch. Based on the threshold fixed on download time, it decides the best protocol to use.

We studied a trace of the access pattern of UB1 collected over 30 hours. We measured the overall amount of data that can be offloaded using our switching algorithm. Our results show that about 16% of the provisioned cloud bandwidth can be saved without degrading the download time service.

Our future plans include the study of the allocation of the cloud's upload bandwidth to the different swarms. We focus on optimizing the usage of cloud resources by studying the relationship between the file size and the swarm characteristics in defining the minimum amount of bandwidth needed to distribute the requested file.

Acknowledgments

This work has been partially funded by the EU in the context of the project *CloudSpaces*: Open Service Platform for the Next Generation of Personal clouds (FP7-317555) and spanish research project DELFIN (TIN2010-20140-C03-03) funded by the Ministry of Science and Innovation.

References

[1] B. Cohen, "Incentives Build Robustness in BitTorrent," 2003.

[2] 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*, May 2013, pp. 327–334.

[3] 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*, ser. DADC '08. New York, NY, USA: ACM, 2008, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/1383519.1383526

[4] S. Priyanka, R. Kalpana, and M. Hemalatha, "Reducing upload and Download Time on Cloud using Content Distribution Algorithm," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 1, pp. 101–105, 2013.

[5] R. Wartel, T. Cass, B. Moreira, E. Roche, M. Guijarro, S. Goasguen, and U. Schwickerath, "Image distribution mechanisms in large scale cloud providers," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 112–117.

[6] 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*, 2010, pp. 567–574.

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

[8] 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*, Sept 2013, pp. 1–2.

[9] X. Leon, R. Chaabouni, M. Sanchez-Artigas, and P. Garcia-Lopez, "Smart cloud seeding for bittorrent in datacenters," *Internet Computing, IEEE*, vol. 18, no. 4, pp. 47–54, July 2014.

[10] B. Wei, G. Fedak, and F. Cappello, "Scheduling independent tasks sharing large data distributed with bittorrent," in *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, Nov 2005, pp. 8 pp.–.

[11] 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*, Nov 2006, pp. 1–11.

[12] 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*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 481–494. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398827

[13] 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*, ser. SIGCOMM '04. New York, NY, USA: ACM, 2004, pp. 367–378. [Online]. Available: http://doi.acm.org/10.1145/1015467.1015508

[14] "Ubuntu one file services," http://one.ubuntu.com/.

[15] 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*. ACM, 2013, pp. 205–212.