Institut Eurécom
Departement of Corporate Communications
2229, route des Cretes
B.P. 193
06904 Sophia Antipolis
FRANCE

Research Report RR-04-098

Cooperative Strategies for File Replication in P2P Networks

October 2004

Anwar Al Hamra

Tel: (+33) 4 93 00 26 26
Fax: (+33) 4 93 00 26 27
Email: alhamra@eurecom.fr

[1] Institut Eurécom's research is partially supported by its industrial members: Bouygues Télécom, Fondation d'entreprise Groupe Cegetel, Fondation Hasler, France Télécom , Hitachi, ST Microelectronics, Swisscom, Texas Instruments, Thales

Cooperative Strategies for File Replication in P2P Networks

Anwar Al Hamra

Institut Eurecom
Departement of Corporate Communications
B.P. 193
06904 Sophia Antipolis, FRANCE
http://www.eurecom.fr
alhamra@eurecom.fr

Abstract. In this report we study the use of P2P networks for file replication. Existing solutions for this service can be largely classified into tree-based and mesh-based approaches. Our first contribution here is a comparison between the scaling behavior of both kinds of approaches. Throughout this comparison, we prove that mesh approaches can be at least as efficient as tree ones.
Our second contribution is a complete analysis of mesh approaches where we identify the main parameters that influence their performance. Our results and conclusions provide new insights for the design of new cooperative architectures.

1 Introduction

File replication in P2P networks has attracted a lot of interest lately especially after the great success of some approaches like *BitTorrent*. Recent statistics by *AlwaysOn Network* magazine [1] prove that *BitTorrent* alone generates around 53% of the total P2P traffic, which in turn accounts for $70 - 80\%$ of the overall European Internet traffic [7].
Existing approaches for file replication can be broadly classified into tree-based and mesh-based approaches, according to the way clients are organized in the network. This report has two main goals. The first one is to compare tree-based and mesh-based approaches. Biersack et al. [2] have analyzed the performance of different tree approaches, a linear chain (*Linear*), a simple tree with an outdegree k ($Tree^k$), and a forest of parallel trees ($PTree^k$). $PTree^k$ splits the file into k parts and sends each part on a distinct tree. A client is an interior node in a tree and a leaf node in the remaining ones. In their analysis, the authors proved that tree-based approaches can be highly efficient; the number of clients served scales exponentially in time for $Tree^k$ and $PTree^k$. In this report we demonstrate that mesh approaches can provide as low service time as tree approaches while being more flexible and more robust against bandwidth fluctuations and client departures. To do so, we introduce two cooperation strategies between peers[1], namely *Least Missing* and *Most Missing*. We evaluate these two strategies analytically as well as through simulations. Our results prove that the *Least Missing* architecture achieves a similar performance to $Tree^k$ while benefiting from more flexibility and simplicity. Also, *Most Missing* scales even better than $PTree^k$ while avoiding the penalty of constructing parallel trees.
To the best of our knowledge, there has been scarcely any comparison between the scaling performance of tree approaches and mesh approaches. We are only aware of one paper by Yang et al. [11] that models the service capacity of mesh approaches. In this paper, the authors prove that such approaches can scale exponentially in time but, no particular architecture was given.

The second goal of this report is an analysis of the scaling behavior of mesh approaches. To this purpose we evaluate extensively the two architectures, *Least Missing* and *Most Missing*, under various assumptions of the system parameters. Based on our results, we present guidelines that help in the design of new architectures depending on the goals to achieve and the deployment scenario.

1.1 Our Contribution

In this report we introduce two mesh cooperative architectures, *Least Missing* and *Most Missing*. The performance of this type of architectures depends on three key designs:

[1] Keep in mind that we use the term peer only in case we want to refer to both, clients and server at the same time.

– **Peer selection strategy:** which among our neighboring peers will we actively trade with, i.e. serve chunks to or request chunks from? In this report, we assume that the file is split into $|\mathcal{C}|$ chunks of equal size.
– **Chunk selection strategy:** which chunk will we preferably serve to, or request from, other peers?
– **Network degree:** what are the optimal indegree/outdegree of peers that achieve a high connectivity between peers and speed up the file dissemination? We define the indegree (respectively outdegree) of a peer as the maximum number of concurrent download (respectively upload) connections at that peer.

As for the peer selection strategy, there are two intuitive but extreme solutions. The first one, referred to as *Least Missing*, is to quickly create a few sources for the entire file, which in turn serve the file and create other sources and so on. The opposite way is to fast upload few chunks to a lot of clients, which allows them to quickly participate into the file delivery. We call this strategy the *Most Missing* one. In both strategies we require peers to schedule the rarest chunks first. Rarest chunks are the least duplicated chunks in the system. However, the motivation behind the choice of these two strategies is not only because they are simple and intuitive. In fact, *Least Missing* and *Most Missing* are just samples that validate how efficient mesh-based approaches can be. We analytically prove that *Least Missing* can simulate a tree distribution (i.e. $Tree^k$) if we choose correctly the indegree and outdegree of peers. We also demonstrate that *Most Missing* can be even more efficient than $PTree^k$, the optimal tree architecture as found in [2]. Moreover, these two strategies help us to understand the main restrictions as well as the power of cooperative distribution architectures without complicating the analysis. Through intensive simulations, we isolate the main parameters that can highly affect the system performance, namely the arrival process of clients, the network degree, the life time of clients, and the upload and download capacity of peers. A key result is that *Most Missing* minimizes the impact of these parameters while the behavior of *Least Missing* varies significantly from one scenario to another. Furthermore, we discuss the importance of the chunk selection strategy. We assess the performance of the two architectures in the case where peers schedule chunks at random instead of rarest ones. Our conclusion is that random selection of chunks produces a high degradation in the system performance.

Note that there are also technical parameters that might have a significant impact on the system behavior. One parameter is the available bandwidth in the network. In this report we assume that peers have limited upload/download capacities but the network is assumed to have infinite bandwidth. The reason is that we want to focus on the advantages and the shortcomings related to our architectures and not to external factors. Another important parameter is the "data management". Previous work [3, 4, 10, 11] has advised to split the file into various "chunks", which permits concurrent downloading from multiple peers. In addition, with this technique, instead of waiting to finish to download the whole file, as soon as a client finishes downloading a chunk, it can start serving it. Yet, the choice of the number and the size of the chunks is critical: the larger the number and the smaller the size of the chunks, the faster the clients participate into the file delivery, which in turn improves the system performance. However, this improvement would be at the cost of a higher overhead induced by more messages exchanged between clients. So, the service provider can choose the appropriate values (i.e. number and size of the chunks) based upon the required goal. We consider here a common chunk size of 256 KB and a number of chunks of $|\mathcal{C}| = 200$, which makes a file of 51.2 MB. We also consider the case of one single file.

To summarize, our goals in this report are: (i) Support the use of mesh approaches rather than tree ones for file replication services and (ii) Present a complete analysis that provides guidelines for the conception of new approaches depending on the scenario and the goals to attempt.

1.2 Notation

Before we go into details, we introduce the parameters that we use in the sequel. We denote by $\mathcal{C}$ and $|\mathcal{C}|$ respectively the set and number of all chunks in the file being distributed. $\mathcal{D}_i$ and $\mathcal{M}_i$ represent the set of chunks that client i has already downloaded and is still missing respectively (with $\mathcal{M}_i \cup \mathcal{D}_i = \mathcal{C}$ and $\mathcal{M}_i \cap \mathcal{D}_i = \emptyset$). Similarly, $d_i \triangleq \frac{|\mathcal{D}_i|}{|\mathcal{C}|}$ and $m_i \triangleq \frac{|\mathcal{M}_i|}{|\mathcal{C}|}$ correspond to the proportions of chunks that client i has already downloaded and is still missing, respectively. The parameter S_{up} stands for the upload capacity of the server while C_{up} and C_{down} represent respectively the upload and download capacity of clients. The indegree of peers is represented by the parameter P_{in} and the outdegree by P_{out}. *One round* or *one unit of time* is the time needed to download the file at rate r, where r is a parameter expressed in Kbps. It follows that, for a file that comprises $|\mathcal{C}|$ chunks, $\frac{1}{|\mathcal{C}|}$ unit of time is needed to download one single chunk at rate r. Finally, the life time (*Life*) of a client, expressed in min, denotes the time the client stays online after it has completely downloaded the file. For example, $Life = 0$ means that the client is selfish and disconnects straight after it finishes the download. Table 1 summarizes all the aforementioned parameters.

Parameter	Definition
$\mathcal{C}$	Set of all chunks
$\|\mathcal{C}\|$	Number of all chunks
N	Number of clients in the system
$\mathcal{D}_i$	Set of chunks that client i has already downloaded
d_i	Proportion of chunks that client i has already downloaded
$\mathcal{M}_i$	Set of chunks that client i is still missing
m_i	Proportions of chunks that client i is still missing
S_{up}	Upload capacity of the server in Kbps
C_{up}	Upload capacity of clients in Kbps
C_{down}	Download capacity of clients in Kbps
P_{in}	Indegree of peers
P_{out}	Outdegree of peers
r	Parameter expressed in Kbps
One round/*One unit of time*	Download time of the entire file at rate r
Life	Life time of clients in min

Table 1. Definition of the parameters used in this report.

1.3 Organization of this Report

The rest of this report is structured as follows. In section 2 we introduce the basic design of our two architectures, *Least Missing* and *Most Missing*. Section 3 summarizes the scaling behavior of *Linear*, $Tree^k$, and $PTree^k$ that we use in the comparison between tree-based and mesh-based approaches. Such a comparison is performed in section 4. Section 5 describes the experimental setup and the simulations results are given in sections 6 and 7. We discuss open questions in section 8 and we conclude this report in section 9.

2 The Basic Design

In this section we present the basic design of our two cooperative architectures. Each architecture includes a peer selection strategy coupled with a chunk selection strategy. We also point out the importance of the network degree as a key design.

2.1 Peer Selection Strategy

The peer selection strategy defines ”trading relationships” between peers and affects the way the network self-organizes. In our simplified model we assume that (i) A client can communicate with any other client in the network and (ii) Each client knows which chunks the other clients in the system hold. Our results should remain valid in practice given that each client knows about a large enough subset of other clients, e.g. 40 to 120 clients as we observed in *BitTorrent* [8].

When a client has some chunks available and some free upload capacity, it uses a peer selection strategy to locally determine which client it will serve next. In this report, we propose and evaluate two strategies:

- *Least missing:* Preference is given to the clients that have many chunks, i.e., we serve in priority client j with $d_j \geq d_i, \forall\, i$. This strategy is inspired by the SRPT (shortest remaining processing time) scheduling policy that is known to minimize the service time of jobs [9].
- *Most missing:* Preference is given to the clients that have few chunks (new comers), i.e., we serve in priority client j with $d_j \leq d_i, \forall\, i$. The rationale behind this strategy is to evenly spread chunks among all clients to allow them to quickly serve other clients. We expect this strategy to engage clients into the file delivery very fast and keep them busy for most of the time.

These two strategies present two extreme ways to engage clients into the delivery process. We can easily verify that the *Least Missing* strategy tries to create fast a few sources with the whole file, which in turn create other few

sources and so fourth. In contrast, the *Most Missing* strategy seeks to upload rapidly a few chunks to a lot of clients and hence, makes all clients active.
Recall that our goal here is not to recommend these two strategies. We rather use them to achieve the two goals that we stated earlier: (i) Support the use of mesh approaches for file replication and (ii) Present guidelines for the design of new architectures. Moreover, these two strategies pave the way to deduce many of other ones that are adequate for specific deployment scenarios.

2.2 Chunk Selection Strategy

Once the receiving client j is selected, the sending client i performs an algorithm to figure out which chunk to send to client j. The mechanism to select which chunk to be served on an active connection aims at modifying the way the chunks get duplicated in the network. A good chunk selection strategy is a key to achieve good performance. Bad strategies may result in many clients with non relevant chunks. To avoid such a scenario, we require the sending client i to schedule the rarest chunk $C_r \in (\mathcal{D}_i \cap \mathcal{M}_j)$ among those that it holds and the receiving client j needs. Rarity is computed from the number of instances of each chunk held by the clients known to the sender. This strategy is inspired from *BitTorrent* and expected to maximize the number of copies of the rarest chunks in the system.

2.3 Network Degree

Given the peer and chunk selection strategies, one interesting question is how to choose the indegree/outdegree of a client subject to its upload/download capacity. By maintaining k concurrent upload connections, a client i could serve k clients at once and intuitively quickly upload the chunks that it holds. In addition, by serving k different clients simultaneously, the client can fully use its upload capacity and thus, maximize its contribution to the system resources. However, this intuition is not always correct. The upload capacity of client i would be divided amongst the k different connections. The larger the value of k, the lower the bandwidth dedicated to each connection. As a result, a large value of k might slow down the rate at which chunks get replicated in the network. For instance, for a tree distribution, a small outdegree of $k = 2$ is optimal (see $Tree^k$ in section 3.2).

Similarly, one could think of maintaining multiple download connections to improve the throughput of clients. Previous work by Gkantsidis et al. [6] has clearly showed that, in a large deployment of parallel download, more than 5 concurrent download connections does not benefit the clients. In this report we investigate *Least Missing* and *Most Missing* under many assumptions on the value of the indegree (P_{in}) and outdegree (P_{out}) of peers. Note that we assume all peers (clients and the server) to have the same outdegree.

3 Summary of Tree Approaches

Biersack et al. [2] analyze the performance of tree approaches for file replication. Their analysis focuses on three architectures, *Linear*, $Tree^k$, and $PTree^k$. For each of the these architectures, the authors derive an upper bound on the number of served clients within t rounds under the assumptions that (i) All N clients arrive to the system at time $t = 0$ and (ii) All peers have homogeneous bandwidth capacities ($S_{up} = C_{up} = C_{down} = r$). In this section we briefly outline the basic idea and the scaling behavior of each of the three architectures. In fact, *Linear*, $Tree^k$, and $PTree^k$ are needed for the comparison that we perform later on between the tree-based and mesh-based approaches.

3.1 *Linear*: A Linear Chain Architecture

The basic idea of the linear chain, referred to as *Linear*, is quite intuitive. *Linear* organizes clients in a chain: The server uploads the file to client 1, which in turn uploads the file to client 2 and so on. The authors analyze this architecture under the following assumptions:

- The server serves sequentially and infinitely the file at rate r. At any point in time, the server uploads the file to one single client.
- Each client starts serving the file once it receives the first chunk.

The authors consider the case where each client uploads the whole file at rate r to one other client before it disconnects. Thus, each client contributes to the system with the same amount of bytes it receives from the system. At time $t = 0$, the server starts serving a first client. At time $t = \frac{1}{|\mathcal{C}|}$, the first client gets the first chunk and starts serving a second client. Likewise, once the second client receives the first chunk at time $t = \frac{2}{|\mathcal{C}|}$, it starts serving a third client and so on. As a result, one obtains a chain that increases by one client each $\frac{1}{|\mathcal{C}|}$ unit of time and by $|\mathcal{C}|$ clients each round. On the other hand, at time $t = 1$, the server finishes uploading the file to the first client. If there are still clients that are not receiving any chunk (i.e. $|\mathcal{C}| > N$), the server then starts a new chain that increases also by one client each $\frac{1}{|\mathcal{C}|}$ unit of time. This process at the server repeats each round, which generates $t + 1$ chains within t rounds (figure 1). Basing on the above analysis, one could easily verify that the number of served clients

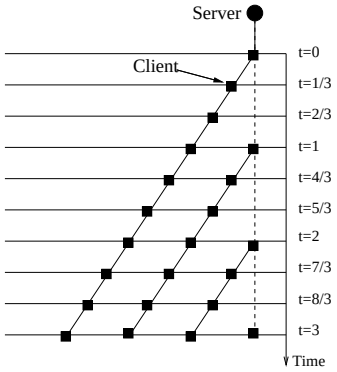


Fig. 1. The evolution of the *Linear* architecture with time. The number of chunks is set to $|\mathcal{C}| = 3$. The black circle represents the server while the black squares represent the clients.

within t rounds is given by:

$$\begin{aligned} N_{Linear}(|\mathcal{C}|, t) &= t + |\mathcal{C}| \cdot t(t-1)/2 \\ &\sim |\mathcal{C}| \cdot t^2 \end{aligned} \tag{1}$$

This result shows that the number of served clients grows linearly with the number of chunks $|\mathcal{C}|$ and quadratically with the number of rounds t. From eq. (1) they derive the time needed (expressed in number of rounds) to serve N clients as follows:

$$\begin{aligned} T_{Linear}(|\mathcal{C}|, N) &= \frac{(|\mathcal{C}| - 2) + \sqrt{(|\mathcal{C}| - 2)^2 + 8 \cdot N \cdot |\mathcal{C}|}}{2 \cdot |\mathcal{C}|} \\ &\sim \frac{|\mathcal{C}| + \sqrt{|\mathcal{C}|^2 + 8 \cdot N \cdot |\mathcal{C}|}}{2 \cdot |\mathcal{C}|} \end{aligned} \tag{2}$$

In their analysis, the authors distinguish three cases depending on the value of the clients to chunks ratio $\frac{N}{|\mathcal{C}|}$:

$$T_{Linear}(|\mathcal{C}|, N) \sim \begin{cases} \frac{1}{2} + \sqrt{\frac{1}{4}} = 1 & \text{for } \frac{N}{|\mathcal{C}|} \ll 1 \\ 2 & \text{for } \frac{N}{|\mathcal{C}|} = 1 \\ \sqrt{\frac{N}{|\mathcal{C}|}} & \text{for } \frac{N}{|\mathcal{C}|} \gg 1 \end{cases}$$

As concerns this architecture, we would like to highlight two main points:

– The main advantage of *Linear* is that it optimizes the transmission time of the file (peers upload and download the file at full rate r). This makes this architecture highly efficient when $\frac{N}{|\mathcal{C}|} \ll 1$.
– However, *Linear* is very slow to engage clients into the file delivery when $\frac{N}{|\mathcal{C}|} > 1$ and thus, clients may wait for too long before they start receiving the file.

Figure 2 shows what we just explained. As long as the number of clients N is smaller than $|\mathcal{C}|$ (i.e. $\frac{N}{|\mathcal{C}|} \ll 1$), the

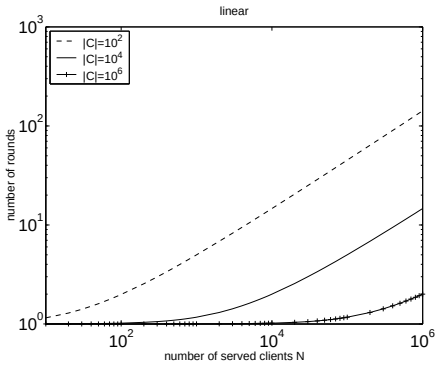


Fig. 2. The service time for *Linear* versus the number of clients N for different values of the number of chunks, $|\mathcal{C}| = \{10^2, 10^4, 10^6\}$.

Linear architecture performs well. When N grows significantly relative to $|\mathcal{C}|$ (i.e. $\frac{N}{|\mathcal{C}|} \gg 1$), the service time can reach extremely large values. For instance, when $|\mathcal{C}| = 10^4$ chunks, one needs about one round to reach $N = 10^2$ clients, 2 rounds for $N = 10^4$, and 15 rounds for $N = 10^6$.

3.2 *Tree*^{k}: A Tree Distribution Architecture

In addition to *Linear*, the authors analyze a simple tree with an outdegree k under the following assumptions:

– The server is located at the root of the tree and uploads the file to k clients in parallel each at rate $\frac{r}{k}$.
– Each client receives the file at rate $\frac{r}{k}$.
– Each interior client in that tree uploads the file to k other clients simultaneously as soon as it receives the first chunk.

Under these assumptions, each interior client contributes k times the amount of bytes it receives from the system while leaf clients upload no chunks at all. Given a download rate of $\frac{r}{k}$, the client needs $\frac{k}{|\mathcal{C}|}$ unit of time to receive a single chunk and k units of time to receive the entire file. At time $t = 0$, the server starts serving the file to k clients, $1, \ldots, k$, each at rate $\frac{r}{k}$. Each of these k clients waits $\frac{k}{|\mathcal{C}|}$ unit of time before it starts serving k new clients. New clients also wait for $\frac{k}{|\mathcal{C}|}$ unit of time before serving the file and so forth. Thereby, each $\frac{k}{|\mathcal{C}|}$ unit of time, one new level is added. Level 0 includes clients $1, \ldots, k$, level 1 includes the k^2 clients served by the k clients at level 0. More generally, the number of clients engaged at level i is k^{i+1} (figure 3).

Given how clients are engaged into the tree and that each client receives the file at rate $\frac{r}{k}$, they derive the number of served clients at time t as:

$$N_{Tree}(|\mathcal{C}|, k, t) \sim k^{\lfloor \frac{(t-k)\cdot|\mathcal{C}|}{k} \rfloor + 1} \tag{3}$$

Eq.(3) shows that the number of served clients scales exponentially with time and with the number of chunks $|\mathcal{C}|$. And the time needed to serve N clients is then:

$$T_{Tree}(|\mathcal{C}|, k, t) = k + \lfloor \log_k \frac{N}{k} \rfloor \cdot \frac{k}{|\mathcal{C}|} \tag{4}$$

By deriving T_{Tree} with respect to k and equating the result to zero, the authors demonstrate that an outdegree $k = 2$ is optimal for $Tree^k$.

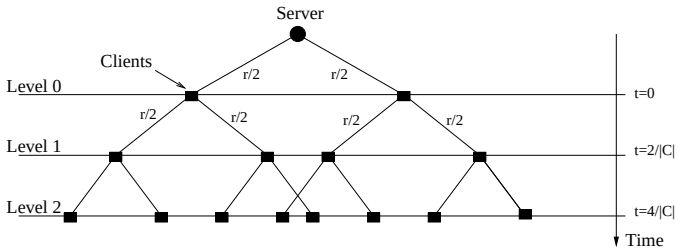


Fig. 3. The evolution of the $Tree^k$ architecture with time for $k = 2$. The client needs two units of time to receive the whole file at rate $\frac{r}{2}$. The black circle denotes the server while the black squares denote the clients.

3.3 *PTree*^{k}: An Architecture Based on Parallel Trees

The overall performance of the tree architecture can be greatly improved if one could capitalize the unused upload capacity at the leaves. This would permit clients to download the file at full rate r instead of at rate $\frac{r}{k}$. The $PTree^k$ architecture allows to do this. In $PTree^k$, the file is partioned into k parts of equal size. If the entire file is divided into $|\mathcal{C}|$ chunks, each of the k parts will comprise $\frac{|\mathcal{C}|}{k}$ disjoint chunks. Then, each part P^i is transmitted over a distinct tree T^i. Each of the k trees includes all N clients in the system; a client is an interior client in one tree and a leaf client in the remaining ones.

Figure 4 depicts the basic idea of $PTree^k$ for $k = 2$. In $PTree^k$, each client helps in distributing $\frac{1}{k}$ of the file

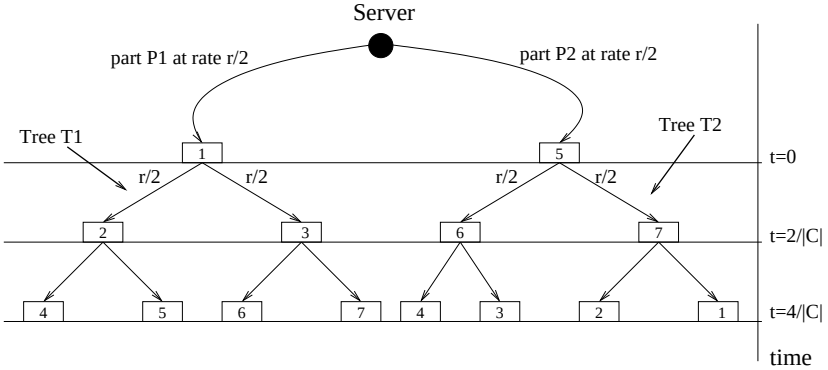


Fig. 4. The evolution of $PTree^k$ with time. The file is divided into two parts ($k = 2$).

(only one part). Given a tree T^i, an interior client serves part P^i to k other clients each at rate $\frac{r}{k}$ and thus, clients upload to the system an amount of bytes equivalent to the full file. The reception of the distinct parts happens in parallel. Each part is received at rate $\frac{r}{k}$ and consequently, all clients can fully use their download capacity. Note that such a distribution architecture was first proposed by Castro et al. [3] under the name of *SplitStream* to increase the resilience against churn (i.e., client departures) in a video streaming application.

$PTree^k$ comprises k trees where each tree includes all clients and consists of $(1 + \lfloor \log_k N \rfloor)$ levels and a height of $\lfloor \log_k N \rfloor$. Throughout each tree T^i, part P^i propagates at rate $\frac{r}{k}$ and thereby, each level induces a delay of $\frac{k}{|\mathcal{C}|}$ unit of time. This means that all parts will be completely received by all N clients at time

$$T_{PTree}(|\mathcal{C}|, k, N) = 1 + \lfloor \log_k N \rfloor \cdot \frac{k}{|\mathcal{C}|} \tag{5}$$

This result is very important as it shows that all clients finish downloading the file at the same time. From the above analysis, the number of served clients within t rounds as:

$$N_{PTree}(|\mathcal{C}|,k,t) = \frac{k^{(\lfloor (t-1)\cdot \frac{|\mathcal{C}|}{k} \rfloor +1)} - 1}{k-1} \sim k^{(\lfloor (t-1)\cdot \frac{|\mathcal{C}|}{k} \rfloor)} \quad (6)$$

As in $Tree^k$, the number of served clients increases exponentially in time and with the number of chunks $|\mathcal{C}|$. The main advantage of $PTree^k$ as compared to $Tree^k$, is that clients receive the file at full rate r instead of $\frac{r}{k}$. Note that, the optimal performance for $PTree^k$ corresponds to $k = 3$.

A comparison between the three architectures is given in table 2 where $PTree^k$ exhibits the best performance. The high efficiency of $PTree^k$ is due to the fact that it engages clients very fast into the file delivery and keeps

Distribution architecture	**Number of served clients**	**Service time**
Linear	$\|\mathcal{C}\| \cdot t^2$	$\frac{(\|\mathcal{C}\|-2)+\sqrt{(\|\mathcal{C}\|-2)^2+8\cdot N\cdot \|\mathcal{C}\|}}{2\cdot \|\mathcal{C}\|}$
$Tree^k$	$k^{\lfloor \frac{(t-2)\cdot \|\mathcal{C}\|}{2} \rfloor +1}$	$k + \lfloor \log_k \frac{N}{k} \rfloor \cdot \frac{k}{\|\mathcal{C}\|}$
$PTree^k$	$k^{(\lfloor (t-1)\cdot \frac{\|\mathcal{C}\|}{k} \rfloor)}$	$1 + \lfloor \log_k N \rfloor \cdot \frac{k}{\|\mathcal{C}\|}$

Table 2. Summary of the scaling behavior of *Linear*, $Tree^k$, and $PTree^k$.

them active most of the time.

4 Mesh approaches versus Tree approaches

As stated in the introduction, one of the two goals of this report is to demonstrate that mesh approaches can be at least as efficient as tree ones. Despite their simplicity, the two architectures *Least Missing* and *Most Missing* permit to do so. The analysis presented in this section shows that (i) *Least Missing* can simulate a tree distribution as with $Tree^k$ while (ii) *Most Missing* ensures the same keys responsible for efficiency as in $PTree^k$, but in a simpler way.

4.1 A Comparison between *Least Missing* and $Tree^k$

In *Least Missing*, each peer tries to serve first the neighbor that has the largest number of chunks amongst all other neighbors. Keep in mind that peers serve the rarest chunks in priority. Despite its simplicity, the number of served clients with *Least Missing* can scale exponentially in time as in $Tree^k$. The key idea is to set the indegree of peers to $P_{in} = 1$ and the outdegree to $P_{out} = k$. For ease of explanation, let us assume that $k = 2$; we analytically prove that *Least Missing* with $P_{in} = 1$ and $P_{out} = 2$ is equivalent to *Tree*$^{k=2}$. In this analysis we make the same assumptions as in *Tree*$^{k=2}$:

- All peers have equal upload and download capacity $S_{up} = C_{up} = C_{down} = r$, where r is a parameter expressed in Kbps.
- Peers maintain an outdegree $P_{out} = 2$ and an indegree $P_{in} = 1$.
- A client can start serving the file once it receives a first chunk.
- Each client remains online until it delivers twice the number of chunks it receives from the system while the server stays online indefinitely.
- All N clients arrive to the system at time $t = 0$.

At time $t = 0$, the server starts serving two disjoint chunks, C_1 and C_2, to two clients 1 and 2, each at rate $\frac{r}{2}$. The server finishes uploading chunks C_1 and C_2 to these two clients by time $t = \frac{2}{|\mathcal{C}|}$. Given that this policy favors clients that have the largest number of chunks, the server will continue uploading to clients 1 and 2 until they completely download the whole file. On the other hand, at time $t = \frac{2}{|\mathcal{C}|}$, clients 1 and 2 have now each one chunk. So each of them starts serving two new clients, each at rate $\frac{r}{2}$ and consequently, 4 new clients are engaged into the file delivery. Clients 1 and 2 will remain uploading the chunks they have to these 4 new clients. This same process repeats and one new level is added each $\frac{2}{|\mathcal{C}|}$ unit of time. Level i includes 2^{i+1} clients, which are served by the 2^i clients located at level $(i-1)$. On the other hand, each client receives the file at rate $\frac{r}{2}$ and the reception lasts for 2 units of time after the first byte has been received. Hence, the number of clients in the system evolves as in a tree with an outdegree $k = 2$. Figure 5 draws the time at which clients start receiving the file. This above analysis can

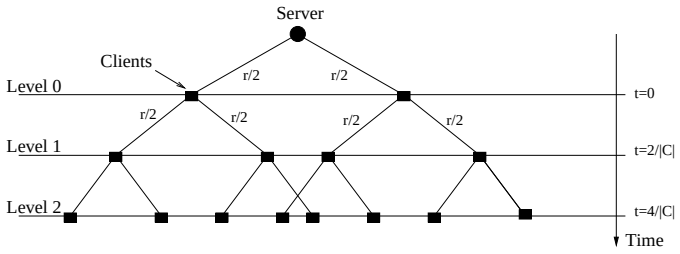


Fig. 5. The evolution of the number of served clients with the *Least Missing* architecture. All N clients arrive to the system at time $t = 0$. Each peer has indegree $P_{in} = 1$ and outdegree $P_{out} = 2$. We assume homogeneous upload/download capacities $S_{up} = C_{up} = C_{down} = b$.

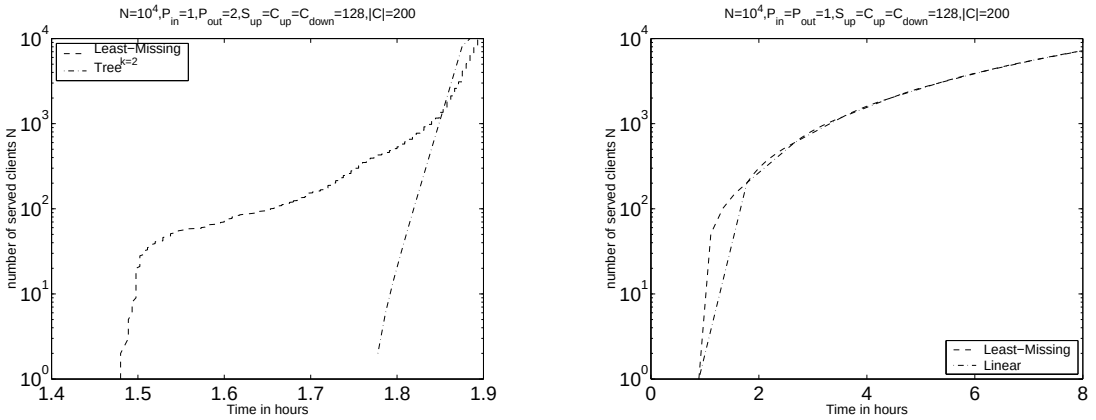
be applied to any integer value of the outdegree k. The key idea is to set $P_{in} = 1$ and $P_{out} = k$. We can then verify that the scenario $P_{in} = P_{out} = 1$ gives a linear chain (*Linear*).

Results To validate our analysis, we compare in figure 6 the evolution of the number of served clients over time for *Least Missing*, $Tree^k$, and *Linear*. The results for *Least Missing* are obtained from simulations[2]. In contrast, the results for *Linear* and $Tree^k$ are computed using respectively equations 1 and 3 in sections 3.1 and 3.2. Figure 6 assumes homogeneous upload and download capacity of peers, $S_{up} = C_{up} = C_{down} = 128$ Kbps. For $P_{in} = 1$ and $P_{out} = 2$, *Least Missing* must act as *Tree*$^{k=2}$ (figure 6(a)). Figure 6(a) shows that the overall time needed to serve 10^4 clients is very close in both approaches. However, the evolution of the number of served clients with time is completely different. We can explain this difference between the analytical and the simulation results as follows. In the analytical model, we assume a perfect synchronization between peers, which is not the case in our simulator. In the simulator, each client is given an id. Consider a connection over which client 1 delivers to client 2 a chunk C_i. Once the chunk is completely delivered, the connection is released and each of the two clients updates its list of chunks. In addition, each of the two clients sorts its neighbors in decreasing order of the number of chunks they hold. Then, each client scans its neighboring list, up to down, until it finds a neighbor to serve. This algorithm is executed sequentially at the client with the lowest id first. This means that, client 1 sorts its neighboring list and starts looking for a new neighbor to serve before client 2 updates the list of chunks it holds. In this case, client 1 would ignore that client 2 has already downloaded chunk C_i and the sorting list of client 1 would not be exact at 100%. As a result, in our simulator, *Least Missing* does not behave as a perfect one and clients that have the largest number of chunks are not necessarily served first, i.e. least missing clients are not always at the top of the neighboring list. This feature has been intentionally added to the simulator to account for the lack of synchronization between peers in real Internet. In practice, when a client receives a chunk, it announces the new chunk to all its neighbors. Thus, a peer may perform its algorithm and choose the new neighbor to serve before all the "announcements" of new chunks have been received.

On the other hand, for $P_{in} = P_{out} = 1$ and under the assumptions of homogeneous upload/download capacities, the *Least Missing* policy should have the same behavior as *Linear* (figure 6(b)). As we can observe from figure

[2] Details about the simulator are given in section 5

6(b), the two results are quite close; the slight difference is once again due to the lack of synchronization between peers.



(a) *Least Missing* vs Tree$^{k=2}$ (b) *Least Missing* vs *Linear*

Fig. 6. The number of served clients against time. All N clients arrive at time $t = 0$. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps.

4.2 *Most Missing* versus $PTree^k$

Having seen how *Least Missing* can scale exponentially with time, we now investigate the scaling behavior of *Most Missing*. In this policy, clients that have the lowest number of chunks are served in priority. Thus, we expect *Most Missing* to engage clients into the file delivery as fast as possible and keep them busy for most of the time. In other words, we believe that *Most Missing* meets the main key features of $PTree^k$. In this section we study the basic behavior of *Most Missing* and we compare it to $PTree^k$. We assume a basic scenario where:

- All peers have equal upload and download capacities, $S_{up} = C_{up} = C_{down} = r$.
- Peers have equal outdegree and indegree $P_{out} = P_{in} = 1$.
- Each client stays online until it receives the whole file and can start serving other clients as soon as it receives a first chunk.
- All N clients arrive to the system at time $t = 0$.

For this scenario, we conduct a simulation with the parameter values given in table 3. In figure 7 we expose

Table 3. Parameter values.

Arrival Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
10^4 at $t = 0$	128 Kbps	128 Kbps	128 Kbps	1	1	0

the number of served clients against time. This figure supports our intuition. *Most Missing* behaves similarly to $PTree^k$; all clients complete very fast and almost at the same time. If we compare the two architectures in absolute terms for the parameter values given in table 3, we find that *Most Missing* needs **3472 seconds** to serve all the 10^4 clients while $PTree^k$ lasts a bit longer, **3584 seconds**. This result shows that we can achieve a high efficiency while avoiding the overhead of constructing parallel trees. Note that the service time achieved by *Most Missing* (i.e. 3472 seconds) is very close to the optimal one; for $S_{up} = C_{up} = C_{down} = 128$ Kbps and for a file of 51.2 MB, the optimal transmission time of the file is 3200 seconds.

By serving always most missing clients, this strategy ensures all clients to progress at almost the same speed. In figure 8 we present a snapshot of the download progress as seen by clients after 45 min of the beginning of the

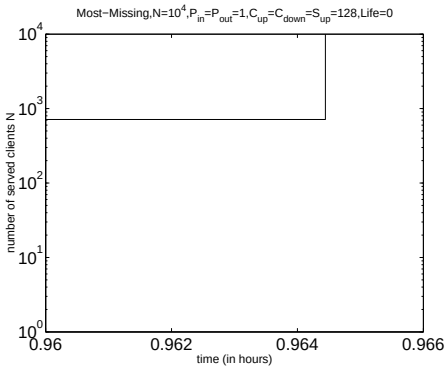


Fig. 7. The number of served clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$).We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

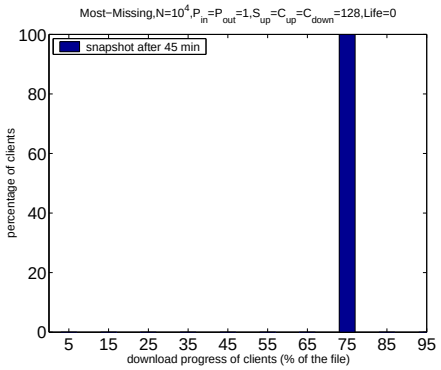
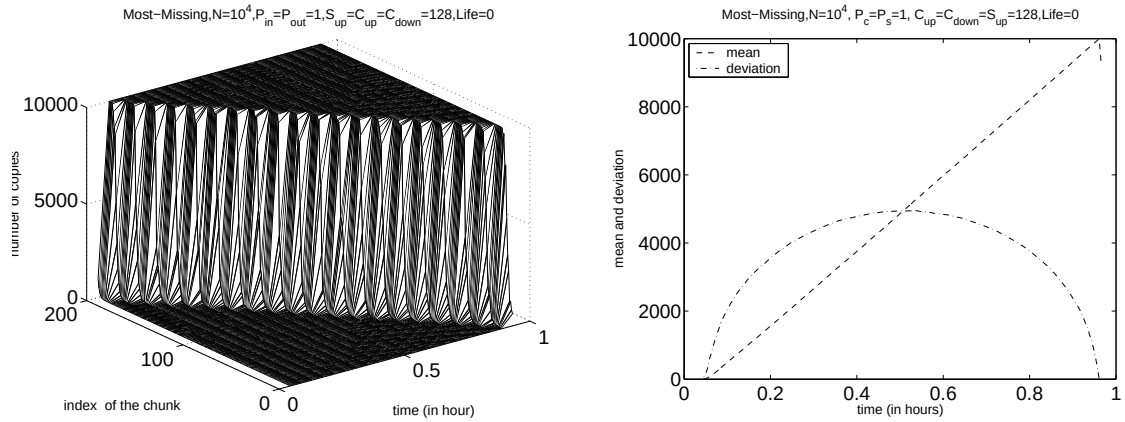


Fig. 8. Snapshot of the download progress of clients for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

simulation. As we can see from this figure, all clients are in the same neighborhood; they are about 75% of the file. For clarity, when we show a snapshot of the download progress, we batch clients that are close in their download within intervals of 10%. For example, if a client has downloaded $X\%$ of the file with $70 \leq X < 80$, we consider that this client has completed 75% of the download.

We further investigate the evolution of the chunks in the network for *Most Missing*. Figure 9(a) draws the number of copies for each chunk as a function of time and the chunk index while the mean and the deviation of the chunks are depicted in figure 9(b). At any time *t*, we compute the mean of the chunks as the sum of the number of copies over all chunks in the system divided by the number of chunks $|\mathcal{C}|$. Figure 9(a) shows that, once a chunk is injected in the system, it gets replicated very fast. As time goes by, the number of copies for each chunk can vary significantly from one chunk to another. This behavior produces a high heterogeneity in the chunks distribution especially after half an hour of simulations where the deviation is maximal (figure 9(b)). To explain the quick replication of chunks with *Most Missing*, we proceed as follows. For the parameter values displayed in table 3, one round is equivalent to 3200 seconds and $\frac{1}{|\mathcal{C}|}$ is equivalent to 16 seconds. For sake of simplicity, we assume that the server starts serving the file at time $t = 0$. The server sends the first chunk C_1 to client 1 at rate r. At time $t = \frac{1}{|\mathcal{C}|}$, client 1 receives completely chunk C_1. Given that the policy tends to serve always rarest chunks to clients that have the fewest number of chunks, at $t = \frac{1}{|\mathcal{C}|}$, the server schedules a new chunk C_2 to a new client 2. Similarly,



(a) Number of chunks vs time and chunk index

(b) Mean and deviation vs time

Fig. 9. The chunks distribution over time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

client 1 starts delivering its chunk C_1 to a new client 3. Let us focus for the moment on chunk C_1. At time $t = \frac{2}{|\mathcal{C}|}$, client 1 downloads completely chunk C_1 to client 3. As a result, at time $t = \frac{2}{|\mathcal{C}|}$, there are two clients (1 and 3) that maintain a copy of the first chunk C_1. By this time $t = \frac{2}{|\mathcal{C}|}$, these two clients deliver that chunk to two new clients and so forth. Hence, the number of copies of chunk C_1 in the system doubles each $\frac{1}{|\mathcal{C}|}$ unit of time. After $\frac{i}{|\mathcal{C}|}$ unit(s) of time, there are 2^{i-1} clients that have the first chunk and only the first chunk. This same analysis can be applied on chunk C_2. Chunk C_2 was injected in the network by the server at time $t = \frac{1}{|\mathcal{C}|}$, i.e. $\frac{1}{|\mathcal{C}|}$ unit of time after the first chunk C_1. By time $t = \frac{i}{|\mathcal{C}|}$, there are 2^{i-2} clients that have chunk C_2 and only chunk C_2. More formally, at time $t = \frac{i}{|\mathcal{C}|}$, chunk j (with $j \leq i$) has 2^{i-j} copies. Figure 10 plots the evolution of the number of copies for the first four chunks versus time. The first chunk C_1 starts with a single copy at time $t = \frac{1}{|\mathcal{C}|}$. At time $t = \frac{2}{|\mathcal{C}|}$, there are 2

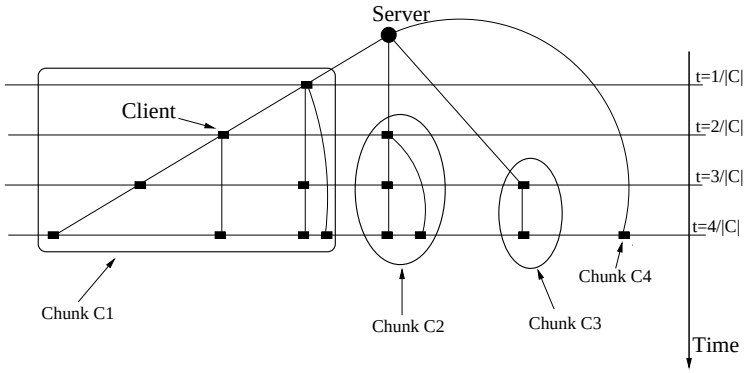


Fig. 10. The theoretical evolution of the number of copies for the first four chunks with time for *Most Missing*. All N clients arrive to the system at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = r$. Clients disconnect as soon as they receive the file (*Life* $= 0$).

copies, and 4 copies at time $t = \frac{3}{|\mathcal{C}|}$.

We outline that this analysis assumes that there are always new clients that have no chunks at all. We refer to clients that hold one chunk as the existing clients and the new clients that hold no chunks at all as the new arrivals. Thus, the above analysis holds true as long as the number of existing clients is less than or equal to the number of new arrivals. And during this period of time, say the start-up period, chunk C_1 will have the largest number of copies, then chunk C_2, and so on. During this start-up period, existing clients are always serving new arrivals and no chunks are exchanged amongst them. Therefore, the number of copies for chunks keeps on growing exponentially. However, once the number of arriving clients becomes less than the number of existing clients in the system, existing clients can start exchanging chunks and it then becomes very hard to keep track how chunks propagate. Yet, figure 9(a) shows that the growth in the number of copies for subsequent chunks is as high as during the start-up period.
In figure 11, we plot the simulation result for the evolution of the number of chunks versus time. We show only

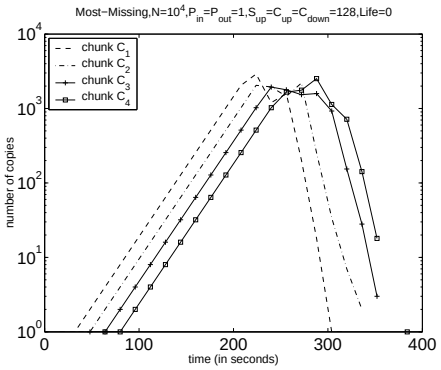


Fig. 11. The simulation result for the evolution of the number of copies of chunks with time. We consider only the first four chunks injected by the server. All N clients arrive to the system at time $t = 0$. Each peer serves one single client at a time. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). Clients disconnect as soon as they receive the file ($Life = 0$).

the first four chunks injected by the server, e.g. chunks C_1, C_2, C_3, and C_4. This figure confirms our analysis. During the start-up period, the number of copies of each of these chunks doubles each $\frac{1}{|\mathcal{C}|}$ unit of time, which is equivalent to 16 seconds in this scenario. In addition, each of these chunks is $\frac{1}{|\mathcal{C}|}$ unit of time late as compared to the antecedent one.

4.3 Preliminary Conclusion

In this section we analyzed the basic behavior of our two architectures. We also presented a comparison between *Least Missing* and $Tree^k$ on one side, and *Most Missing* and $PTree^k$ on the other side. The comparison permitted to validate our intuition; mesh approaches can be more efficient than tree ones while being simpler, more flexible, and more dynamics. Note that this comparison does not take into account the challenges of constructing and maintaining the trees.

We now continue our evaluation of *Least Missing* and *Most Missing* and conduct extensive simulations over a wide set of scenarios. Throughout these simulations, we point out the main parameters that guide the performance of mesh-based approaches. We present two sets of results: The first one is for an instantaneous arrival of all clients at time $t = 0$ while the second set of results assumes a Poisson arrival of clients. Within each set, we start with a basic scenario where we assume (i) $S_{up} = C_{up} = C_{down} = 128$ Kbps, (ii) $Life = 0$ (clients are selfish), and (iii) $P_{in} = P_{out} = 1$. This basic scenario allows us to understand the importance of the way peers organize themselves and cooperate in the system. We then investigate separately the impact of the following parameters:

– *Indegree/Outdegree of peers:* Our goal here is to see whether parallel upload and download of the chunks speed up the replication of the file.

– *Life time of clients:* We study the performance of the system when clients stay for 5 additional min after they completely retrieve the file. Intuitively, having altruistic clients increases the potential service of the system. However, the rational behind this scenario is to point out the conditions to guarantee a good scaling behavior even in worst cases where peers are completely selfish.
– *Upload and download capacity of clients:* We consider the case of clients with heterogeneous bandwidth capacities. We would like to understand how we can prevent clients with low bandwidth capacities from highly damaging the performance of the system.
– *Chunk selection strategy:* We evaluate the two architectures under a random selection of chunks where the goal is to prove that scheduling the appropriate chunks is needed for efficiency. To avoid confusion, we assume that peers schedule always rarest chunks first, unless stated otherwise.

5 Experimental Setup

For the purpose of evaluating our two cooperative architectures, we made use of the simulator developed by Felber et al.[5]. This simulator allows us to observe step-by-step the distribution of large files among all peers in the system according to several metrics.

5.1 Simulation Methodology

The simulator is essentially event-driven, with events being scheduled and mapped to real-time with a millisecond precision. The transmission delay of each chunk is computed dynamically according to the link capacities (minimum of the sender upload and receiver download) and the number of simultaneous transfers on the links (bandwidth is equally split between concurrent connections).

Once a peer i holds at least one chunk, it becomes a potential server. It first sorts its neighboring clients according to the specified peer selection strategy. It then iterates through the sorted list until it finds a client j that (i) Needs some chunks from $\mathcal{D}_i$ ($\mathcal{D}_i \cap \mathcal{M}_j \neq \emptyset$), (ii) Is not already being served by peer i, and (iii) Is not overloaded. We say that a peer (client or server) is overloaded if it has reached its maximum number of connections *and* has less than 128 Kbps bandwidth capacity left. Peer i then applies the chunk selection strategy to choose the rarest chunk to send to client j. Peer i repeats this whole process until it becomes overloaded or finds no other client to serve.

5.2 Deployment Scenarios

We specifically focus on two deployment scenarios that correspond to real-world applications of cooperative content distribution. In the first scenario, we assume that all clients arrive to the system at the same time. This is a crucial scenario that may be the case where (i) A critical data, e.g. anti-virus, must be updated over a set of machines as fast as possible or (i) A flash crowd, i.e. a large number of clients that arrive to the system very close in time.

The second scenario corresponds to the case where clients arrive progressively to the system. In this scenario, the distribution continues over several client ”generations”, with some clients arriving well after the first ones have already left. We model this scenario as a Poisson process with rate λ.

5.3 Metrics

Our simulator allows us to specify several parameters that define its general behavior and operating conditions. The most important ones relate to the content being transmitted (file size, chunk size), the peer properties (arrival process, bandwidth capacities, life times, indegree and outdegree), and global simulation parameters (number of initial servers, simulation duration, peer selection strategy, chunk selection strategy). Table 4 summarizes the values of the main parameters used in our simulations.

We have considered several metrics in our evaluation of the two strategies. We briefly outline below the major properties that we are interested in:

– *Download times:* The duration of the file download as experienced by individual clients. In general, shorter times are better and variance should be minimized.

Parameter	Value
Chunk size	256 KB
Number of chunk $\|\mathcal{C}\|$	200
File size	51.2 MB
Peer arrival rate:	Continuous/Simultaneous
Peer bandwidth (download/upload)	Homogeneous/Heterogeneous
Clients life time	Selfish/Altruistic
Active connections per peer	$1-5$
Number of initial servers	1
Duration of simulation	8 hours
Peer selection strategy	*Least Missing*/*Most Missing*
Chunk selection strategy	Rarest/Random

Table 4. Parameters used in the simulations.

– *Download progress:* The progress of the file download over time seen by each of the clients. In general, regular progress is desirable (i.e., clients should not be stalled for long periods of time).
– *Chunk distribution:* The evolution over time of the frequency of the chunks in the system. The variance of chunk frequencies should be minimized.

6 Simulation Results: Instantaneous Arrival of Clients

6.1 Basic Results

To better understand how the different system parameters can influence the scaling behavior of each of the two architectures, we start with a basic scenario and then extend our analysis and consider more complex ones. In the basic scenario we make the following assumptions:

– All peers (server and clients) have equal upload and download capacities, $S_{up} = C_{up} = C_{down} = r$.
– Each client stays online until it receives the whole file.
– Peers have equal outdegree and indegree $P_{out} = P_{in} = 1$.
– All N clients arrive to the system at time $t = 0$.

Actually, this is the same scenario that we considered in section 4.2 when comparing *Most Missing* to $PTree^k$. Despite its simplicity, this scenario provides important insights into how the performance of the system is influenced by the way peers cooperate in the network. Figure 12 compares *Least Missing* to *Most Missing* for the parameter values given in table 5. This figure exhibits that, overall, *Most Missing* performs much better than *Least Missing*.

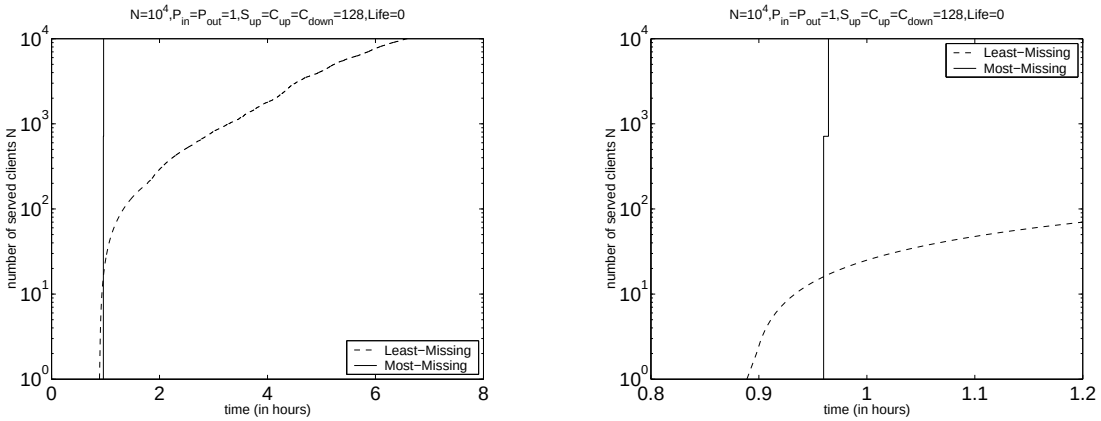
Table 5. Parameter values.

Arrival Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
10^4 at $t = 0$	128 Kbps	128 Kbps	128 Kbps	1	1	0

In absolute terms, *Most Missing* takes **3472 seconds** ($\sim$ 1 hour) to serve 10^4 clients. In contrast, to serve the same number of clients, *Least Missing* needs a larger time, up to **23728 seconds** ($\sim$ 7 hours).

However, from figure 12(b) we can notice that *Least Missing* optimizes the service time of the first few clients, which is also clear from figure 13. In figure 13, we see how *Least Missing* pushes quickly few clients to completion and maintains the majority of clients early in their download.

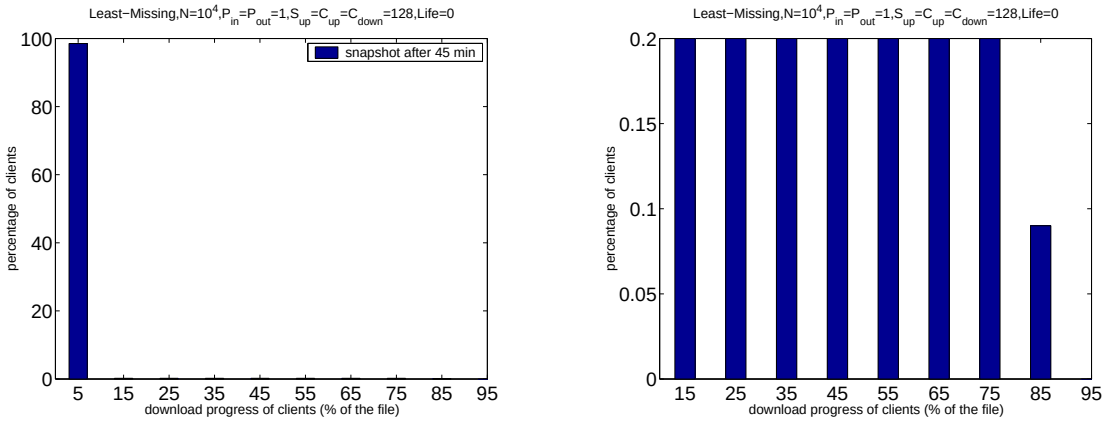
We explain the behavior of *Least Missing* as follows. Theoretically, under the assumptions of $P_{in} = P_{out} = 1$ and $C_{up} = C_{down} = S_{up} = r$, a perfect *Least Missing* policy would result in one single linear chain. Such a chain increases by one client each $\frac{1}{|\mathcal{C}|}$ unit of time. In that chain, the download time of each client is one unit of time. This works as follows. Assume the server starts delivering the file at time $t = 0$ to a first client 1. By time $t = \frac{1}{|\mathcal{C}|}$,



(a) overall view (b) zoom over the first few clients

Fig. 12. The number of served clients against time for *Least Missing* and *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

.



(a) overall view (b) zoom over advanced clients

Fig. 13. Snapshot of the download progress of clients for *Least Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

client 1 receives a first chunk and starts serving a second client 2 and so forth. More formally, client i receives a first byte of the file by time $t = \frac{i-1}{|\mathcal{C}|}$. In addition, this client i will always have one and only one chunk more than client $i + 1$. This means that, the root of the chain, in our scenario client 1, has the largest number of chunks, then client 2, etc. At time $t = 1$, the server finishes uploading the file to client 1. Even though the server becomes free, it does not initiate a new chain. Indeed, we assume here that the client disconnects once it receives the whole file. So, at time $t = 1$, client 1 leaves the network and client 2 is left stranded. In addition, client 2 has the largest number of chunks amongst all other clients in the system. Therefore, at time $t = 1$, the server delivers to client 2 the last chunk this client misses. At time $t = 1 + \frac{1}{|\mathcal{C}|}$, the same process repeats; client 2 disconnects and the server uploads to client 3 the last chunk this client misses. As a consequence, the server will be always delivering a last chunk to the client located at the root of the chain.

As mentioned earlier, there is a lack of synchronization between peers and clients with the largest number of

chunks are not always served first, i.e. a peer may not always serve its successor in the chain. As a result, we obtain multiple chains that progress in parallel and the performance of *Least Missing* is better than that of a linear chain.

If we look at the chunk distribution in figure 14, we can notice that the number of copies of the chunks grows linearly in time.

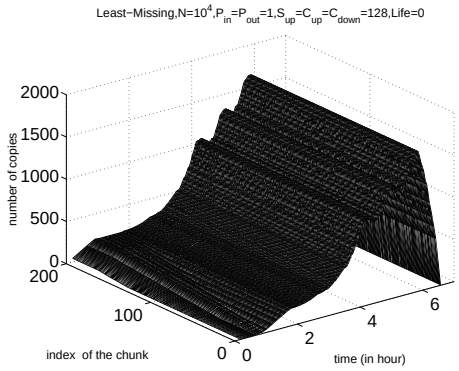


Fig. 14. The chunks distribution over time for *Least Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

Conclusions We compared the two strategies *Least Missing* and *Most Missing* under the same basic and homogeneous scenario. This scenario has clearly shown how the cooperation strategy between peers guides the behavior of the system. For instance, *Most Missing* optimizes the overall service time because it engages rapidly clients into the file delivery and keeps them busy for most of the time. In contrast, *Least Missing* minimizes the delay experienced by the first few clients while the last client to complete notices a high delay. Note that, for $P_{in} = P_{out} = 1$, *Least Missing* consists of multiple linear chains that progress in parallel and is not very efficient when $\frac{N}{|\mathcal{C}|} \gg 1$, which is the case here.

6.2 Impact of the Indegree/Outdegree of Peers

Previous work by Yang et al. [11] studies the service capacity of mesh-based approaches. Their analysis proves that, when all peers have equal bandwidth capabilities and when the network has infinite bandwidth capacity, the optimal growth in the service capacity of the system is for an outdegree of 1. In this section, we address this point and study the impact of the number of incoming/outgoing connections a peer can simultaneously maintain. We allow each peer to have up to $P_{out} = 5$ upload and $P_{in} = 5$ download connections (table 6).

Table 6. Parameter values.

Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	$Life$
$N = 10^4$ at $t = 0$	128 Kbps	128 Kbps	128 Kbps	5	5	0

Most Missing We first analyze the *Most Missing* policy with the number of completed clients graphed in figure 15. The results shown in this figure are in accordance with former ones: Clients download the file quickly and finish at almost the same time. In addition, the chunk distribution over time (figure not shown) exhibits the same tendency as before, i.e. chunks get duplicated at an exponential rate. What is interesting here is that, having multiple download/upload connections is not of benefit to the *Most Missing* policy. This result confirms previous conclusion

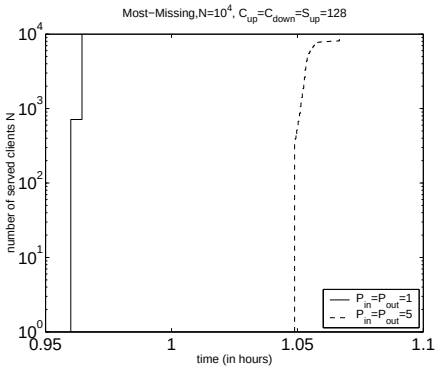
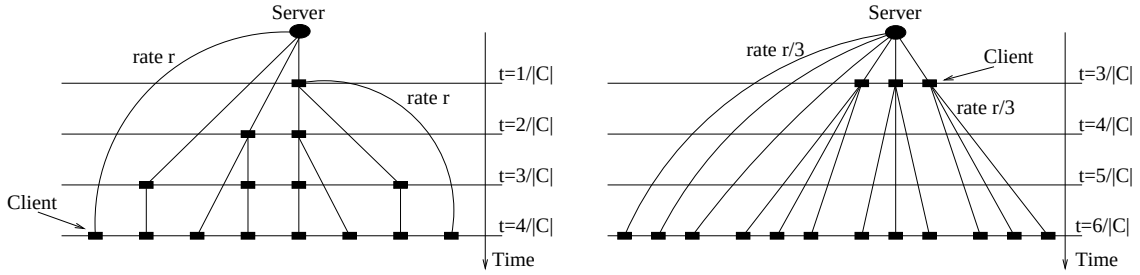


Fig. 15. The number of completed clients against time for *Most Missing*. All N clients arrive at time $t = 0$. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

[11]: In a homogeneous environment and in uncongested bandwidth network, an outdegree/indegree of 1 is optimal. Such a value allows the chunks to double their number of copies each $\frac{1}{|\mathcal{C}|}$ unit of time.

To better understand the impact of the indegree/outdegree of peers on *Most Missing*, we plot in figure 16 the evolution of the number of copies for one single chunk with *Most Missing* for $P_{in} = P_{out} = 1$ and $P_{in} = P_{out} = 3$; when $P_{in} = P_{out} = 1$, the chunk is delivered at full rate r and at rate $\frac{r}{3}$ when $P_{in} = P_{out} = 3$. From figure 16,



(a) $P_{in} = P_{out} = 1$ (b) $P_{in} = P_{out} = 3$

Fig. 16. The evolution of the number of copies of one single chunk in *Most Missing*. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = r$.

we can see that, when $P_{in} = P_{out} = 1$, the chunk can be duplicated over 15 clients within $\frac{4}{|\mathcal{C}|}$ unit of time while we need $\frac{6}{|\mathcal{C}|}$ unit of time when $P_{in} = P_{out} = 3$.

Least Missing The performance of a perfect *Least Missing*, on the other hand, should be independent of the value k of the indegree/outdegree of peers as long as we assume homogeneous scenario with $S_{up} = C_{down} = C_{up} = r$ and $P_{in} = P_{out} = k$. For ease of explanation, we assume peers with an outdegree $k = 2$, i.e. $P_{in} = P_{out} = 2$. At time $t = 0$, the server starts serving two clients 1 and 2 each at rate $\frac{r}{2}$. By time $t = \frac{2}{|\mathcal{C}|}$, clients 1 and 2 obtain each a first chunk. Now, client 1 can start serving its chunk to two clients. Given that the policy is *Least Missing* first, client 1 seeks for clients that hold the largest number of chunks and that have free download capacity. One candidate is client 2. Actually, at $t = \frac{2}{|\mathcal{C}|}$, only clients 1 and 2 hold each one chunk while other clients hold no chunks at all. Thereby, client 1 uploads its chunk to client 2. Client 1 can still have one more upload connection and a new client 3 is then served. The upload capacity of client 1 is equally divided amongst clients 2 and 3. This makes the overall download rate of client 2 equal to r and thus, the time client 2 takes to completely download the

file is one round[3]. Similarly, client 2 uploads its chunk to client 1 and to a new client 4 each at rate $\frac{r}{2}$. As a result, at time $t = \frac{2}{|\mathcal{C}|}$, two new clients start receiving the file. Note that the server will be always uploading to clients 1 and 2, which in turn will be always uploading to clients 3 and 4 (see figure 17). As a result, we obtain two chains

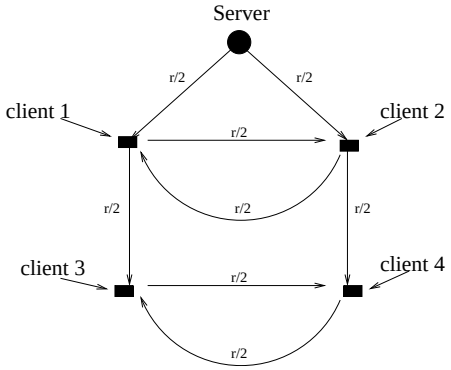


Fig. 17. The theoretical evolution of *Least Missing* for $P_{in} = P_{out} = 2$. We focus only on the first four clients in the system.

that increase each by one client every $\frac{2}{|\mathcal{C}|}$ unit of time. This would give us the same result as if we had one single chain that increases by one client each $\frac{1}{|\mathcal{C}|}$ unit of time, which corresponds to $k = 1$. This analysis can be applied to any integer value of k and therefore, *Least Missing* with $k = 5$ is equivalent to *Least Missing* with $k = 1$.

We recall that the above analysis holds true only for a perfect *Least Missing*. However, the simulation results show completely different tendency. Figure 18 plots the number of completed clients versus time. As we can

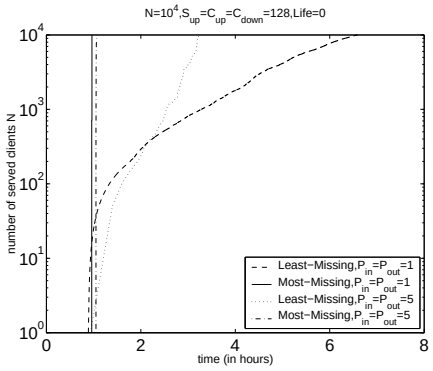
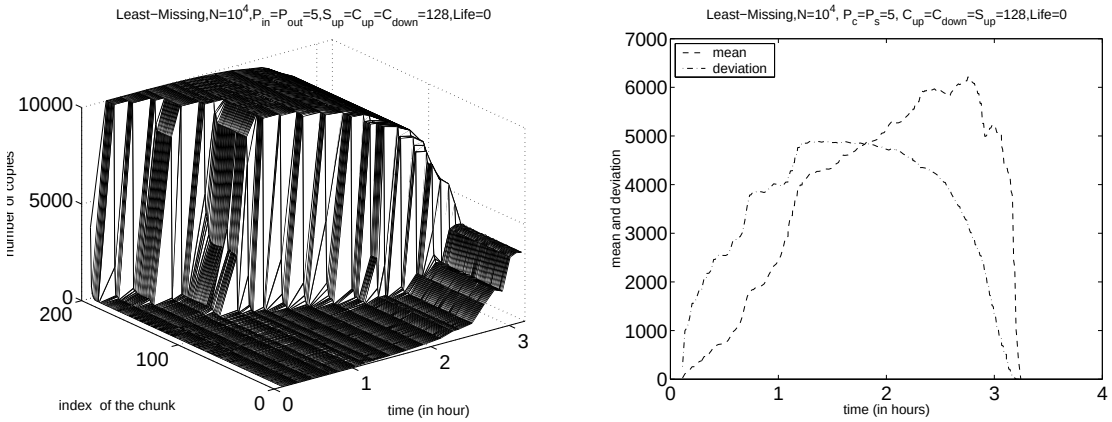


Fig. 18. The number of completed clients against time. All N clients arrive at time $t = 0$. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

conclude from this figure, for $P_{in} = P_{out} = 5$, the time needed to serve 10^4 clients is halved as compared to the scenario where $P_{in} = P_{out} = 1$; **11680 seconds** instead of **23728 seconds**.

Moreover, from figure 19, we find that the expansion of the number of copies for one of the first 5 chunks, say chunk C_1, injected by the server at time $t = 0$, follows a tree with an outdegree $k = 5$. For instance, within 404 seconds, that chunk C_1 has 3667 copies. In a tree with an outdegree $k = 5$, with 5 levels, we can reach up to 3906

[3] Client 2 maintains two download connections at rate $\frac{r}{2}$ each. One connection to the server and the second one to client 1.



(a) Number of chunks vs time and chunk index (b) Mean and deviation vs time

Fig. 19. The chunks distribution over time for *Least Missing*. All N clients arrive at time $t = 0$. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Each peer maintains at most 5 download and 5 upload connections ($P_{in} = P_{out} = 5$). Clients disconnect as soon as they receive the file (*Life* $= 0$).

copies within 400 seconds. While the analysis gets complicated for subsequent chunks, our intuition is that, as the outdegree of peers increases, *Least Missing* engages clients faster into the file delivery and its performance gets better. To validate our intuition, we run a new set of simulation with incoming and outgoing connections of peers of $P_{in} = P_{out} = 3$. What we would like to see is that the number of completed clients is somewhere between that of ($P_{in} = P_{out} = 1$) and that of ($P_{in} = P_{out} = 5$), which is the case in figure 20.

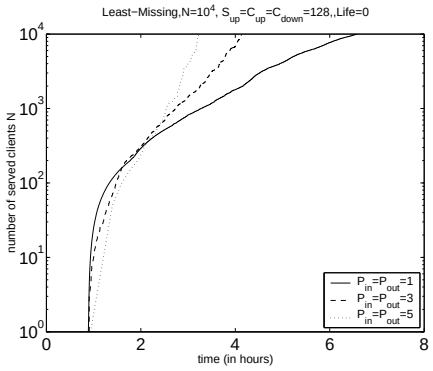


Fig. 20. The number of completed clients against time for *Least Missing*. All N clients arrive at time $t = 0$. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

Conclusions In this section we investigated the impact of the network degree on our two architectures. As P_{in} and P_{out} go from 1 to 5, the performance of *Most Missing* slightly degrades while the performance of *Least Missing* doubles. Even though, we argue that parallel download and upload of the chunks can offer many advantages in real environments. Mainly, it allows clients to fully use their upload and download capacities, which makes the system more robust against bandwidth fluctuations in the network and client departures. In addition, parallel connections ensures a good connectivity between peers in the system.

6.3 Impact of the Life Time of Clients

The way clients behave in the network has a high impact on the system performance. So far we have focused on the case of selfish clients that disconnect once they are done. However, in real Internet we expect to observe both kind of clients, selfish and altruistic. Actually, we do not expect clients to stay intentionally to help into the file distribution, but just because not all of them monitor with high attention their download progress and disconnect once it is finished. In this section we assume that clients stay online for 5 additional min (*Life* $= 5$). Other parameter values that we consider in this section are given in table 7.

Table 7. Parameter values.

Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
$N = 10^4$ at $t = 0$	128 Kbps	128 Kbps	128 Kbps	1	1	5

Most Missing As in previous sections, we start with the *Most Missing* strategy. In this strategy, we have seen that, under the assumption of instantaneous arrivals, all clients finish at almost the same time. Thus, clients need not to stay in the system after they complete their download simply because there remain no clients to be served and consequently, *Most Missing* is very insensitive to the parameter *Life* (figure 21).

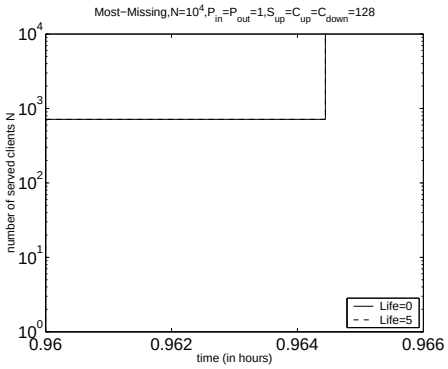


Fig. 21. The number of completed clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connections ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps.

Least Missing Let us now see what happens with *Least Missing* in case we allow clients to stay and help with the file delivery. We have seen how this strategy serves clients progressively over several hours. When we assume altruistic clients with *Life* $= 5$, the system will have a greater potential service and its performance increases. From figure 22 we can see that the performance of *Least Missing* becomes much better; **9920** seconds to serve $N = 10^4$ clients instead of **23728** seconds for *Life* $= 0$. From figure 23, as compared to the case with *Life* $= 0$ (figure **??**) on the other hand, we can see that the chunk distribution becomes steeper, i.e. chunks get replicated faster in the system.

Conclusions In this section we saw that the life time of clients can significantly influence the performance of the system when all clients arrive to the system at time $t = 0$. This is the case for the architectures that serve clients sequentially like *Least Missing*. In contrast, for strategies like *Most Missing* where clients finish at almost the same time, this parameter has no effect. Thus, to minimize the influence of the behavior of clients, the system must engage clients into the file delivery very early and hold them in the system as long as possible.

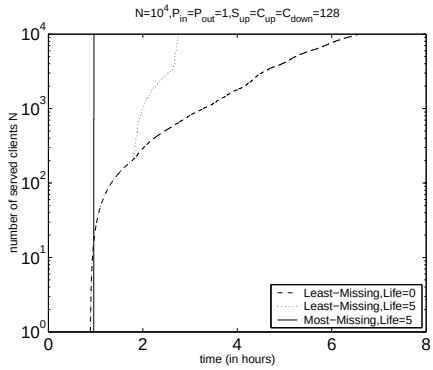


Fig. 22. The number of completed clients against time. All N clients arrive at time $t = 0$. Each peer maintains at most 1 download and 1 upload connections ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps.

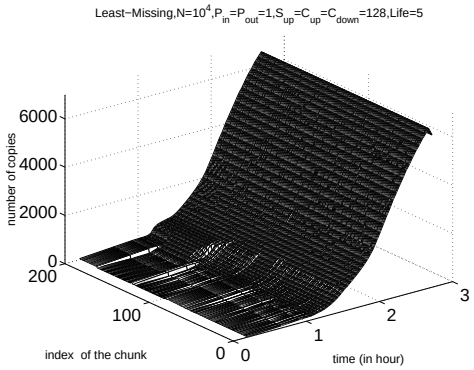


Fig. 23. The chunks distribution over time for *Least Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most 1 download and 1 upload connections ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients stay online 5 additional min after they have received the file ($Life = 5$).

6.4 Impact of the Bandwidth Heterogeneity of Clients

So far, we have assumed peers with homogeneous bandwidth capacities. However, in real Internet, the server has more upload capacity than clients. In addition, most ISPs provide clients with asymmetric bandwidth capacities; usually, the download capacity of a client is larger than its upload capacity. Moreover, clients typically have heterogeneous bandwidth capacities, ranging from dial-up modems to broadband access. In this section we account for this heterogeneity and consider two classes of clients: 50% with $C_{up} = 128$ Kbps and $C_{down} = 512$ Kbps and the other 50% with $C_{up} = 64$ Kbps and $C_{down} = 128$ Kbps. The server has a higher capacity of $S_{up} = 1024$ Kbps. The values that we consider here are just to give new insights into how the heterogeneity of clients can change the performance of the system. Table 8 outlines the remaining parameter values.

Table 8. Parameter values.

Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	$Life$
$N = 10^4$ at $t = 0$	$128/64$ Kbps	$512/128$ Kbps	1024 Kbps	1	1	0

Most Missing When we assume two sets of clients with different upload capacities, we expect clients to progress uniformally. The reason is that, on average, all clients receive a similar service; each client is served half of the time by clients that have an upload of 128 Kbps and the other half of time by clients with 64 Kbps. As a result, clients with low upload capacity would delay the download progress of the clients with the higher upload capacity. However, what we learn from figure 24 is completely different. Figure 24 shows two batches of clients: The

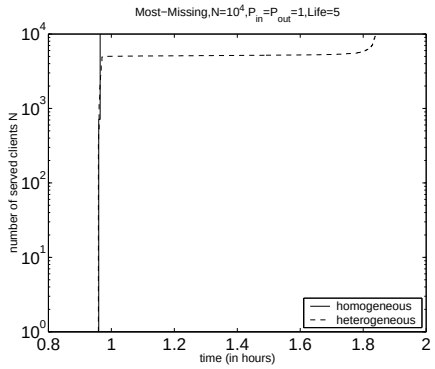


Fig. 24. The number of completed clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). The upload and download capacities are $S_{up} = 1024$ Kbps, $C_{up} = 128/64$ Kbps, and $C_{down} = 512/128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).

first batch of clients completes the download after **3472** seconds of simulations while the second batch finishes after **6636** seconds. The first batch of clients corresponds to clients that have the higher bandwidth capacities ($C_{up} = 128$ Kbps and $C_{down} = 512$ Kbps) while the second batch represents clients that provide $C_{up} = 64$ Kbps and $C_{down} = 128$ Kbps. This result is extremely important as it shows that *Most Missing* offers to its clients a service proportional to their upload capacity. This is a desirable property as clients that upload the best must receive the best service. If we observe the download progress of clients after 45 min of simulations, we notice clearly two big batches of clients, the first one with the large bandwidth capacity at about 75% of the file while the second one at almost half the distance (figure 25). This above behavior of *Most Missing* is not really intuitive.

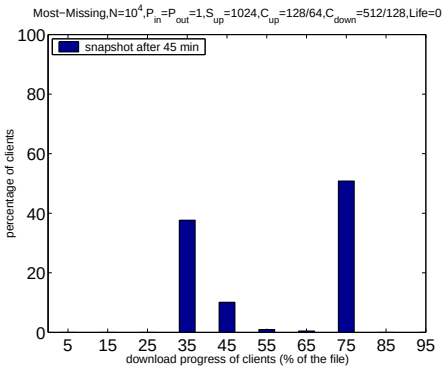


Fig. 25. The download progress of clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). The upload and download capacities are $S_{up} = 1024$ Kbps, $C_{up} = 128/64$ Kbps, and $C_{down} = 512/128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).

We can imagine that *Most Missing* sorts clients by classes according to their upload capacities and only clients that belong to the same class cooperate among each others. Indeed, we believe that such a behavior is somehow influenced by the properties of our simulator. Basically, the condition $P_{in} = P_{out} = 1$ means that each peer has at most one download and one upload connection. However, to avoid having unused bandwidth, we allow a peer to have additional download (or upload) connections provided it has at least 128 Kbps of free download (or upload) bandwidth. Thus, the clients that have a high upload/download capacity would benefit from having more download connections and will progress faster than the other ones.

Finally, if we look at the evolution of the number of chunks in figure 26, we notice a sudden fall that is due to the departure of the first batch of clients. Even though, the number of chunks in the system keeps on expanding fast.

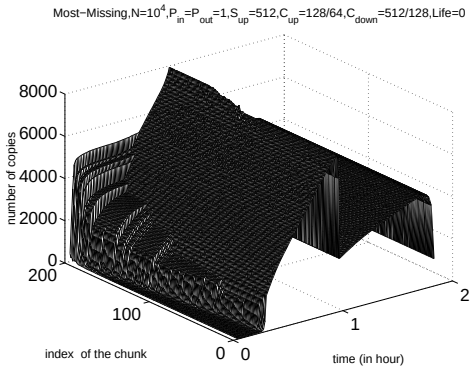


Fig. 26. The chunks distribution over time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). The upload and download capacities are $S_{up} = 1024$ Kbps, $C_{up} = 128/64$ Kbps, and $C_{down} = 512/128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).

Least Missing As we have seen so far, the *Least Missing* policy is similar to a linear chain. Therefore, we expect this policy to behave very badly in a heterogeneous environment like the one we consider in this section. The reason is that clients are served sequentially, which means that the rate at which chunks propagate in the system is highly affected by the lowest upload capacity of peers. Figure 27 confirms what we just mentioned. From this figure, we can see how the performance of *Least Missing* has significantly dropped and, in contrast to *Most Missing*, the service of clients with high bandwidth capacity is highly damaged by the low upload capacity of other clients. If we take a closer look at figure 27, we notice a very short service time of the first client with *Least Missing*, around 800 seconds. Our explanation is that this client, with for sure a high download capacity of $C_{down} = 512$ Kbps, was the first to join the server and received the file at full download rate. We can easily verify that, receiving 200 chunks of 256 KB each over a connection of 512 Kbps takes 800 seconds. Overall, we can conclude the same tendency for *Least Missing*; few clients progress fast while the majority do slowly (figure 28).

What is new for *Least Missing* is that this scenario reveals a higher number of chunks in the system (figure 29) as compared to the basic scenario with homogeneous bandwidth capacities ($S_{up} = C_{up} = C_{down} = 128$ Kbps, figure 14). But this does not mean a larger potential service capacity because most of existing chunks are shared by almost all clients, and these clients are waiting for new chunks to be injected by the server.

Conclusions In this section we addressed the influence of the bandwidth heterogeneity of clients on the system performance. We saw how clients with low bandwidth links can slow down significantly the dissemination of the file when clients are served sequentially, which is the basic tendency of *Least Missing*. In such an architecture, clients may wait for too long to receive new chunks. In contrast, *Most Missing* leverages better the heterogeneity of clients. This strategy keeps all peers working most of the time, which ensures a high service capacity of the system and consequently, allows clients to progress fast.

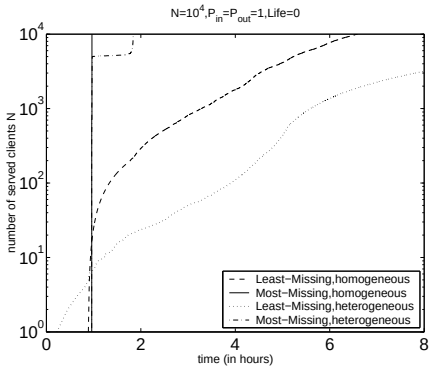
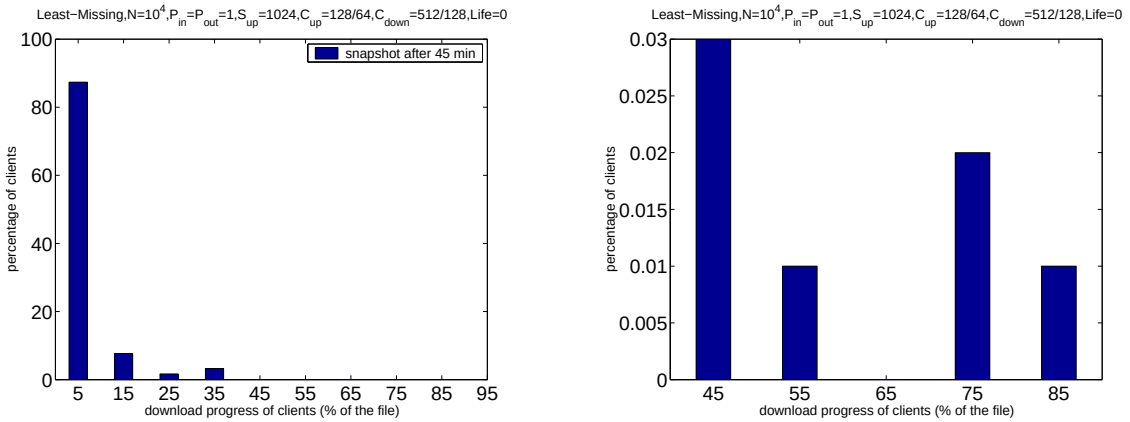


Fig. 27. The number of completed clients against time. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). Clients disconnect as soon as they complete their download (*Life* $= 0$).



(a) overall view (b) zoom over advanced clients

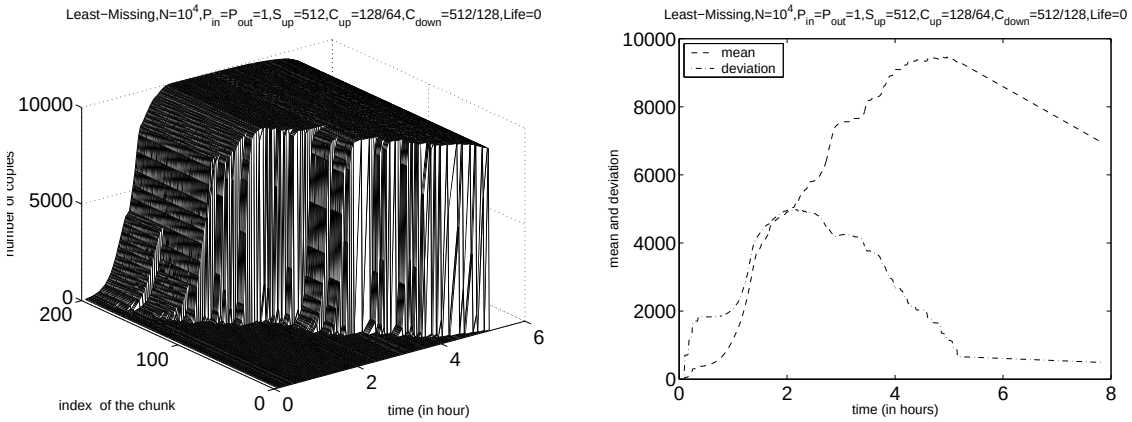
Fig. 28. Snapshot of the download progress of clients for *Least Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). The upload and download capacities are $S_{up} = 1024$ Kbps, $C_{up} = 128/64$ Kbps, and $C_{down} = 512/128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

6.5 Impact of the Chunk Selection Strategy: Random rather than Rarest

In their basic version, *Most Missing* and *Least Missing* require peers (server and clients) to serve rarest chunks first, i.e. the least duplicated chunks in the system. In this section we investigate a possible simplification of the system. We allow peers to schedule chunks at random as follows. The sending peer i selects a chunk $C_i \in (\mathcal{D}_i \cap \mathcal{M}_j)$ at random among those that it holds and the receiving client j needs. Under this assumption, we refer to *Most Missing* as *Most Missing Random* and to *Least Missing* as *Least Missing Random*. Our goal through *Most Missing Random* and *Least Missing Random* is to see whether this feature can be integrated to the system without sacrificing a lot of performance. To this purpose, we run new simulations with the parameter values given in table 9.

Table 9. Parameter values.

Arrival Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
10^4 at $t = 0$	128 Kbps	128 Kbps	128 Kbps	1	1	0



(a) Number of chunks vs time and chunk index (b) Mean and deviation vs time

Fig. 29. The chunks distribution over time for *Least Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). The upload and download capacities are $S_{up} = 1024$ Kbps, $C_{up} = 128/64$ Kbps, and $C_{down} = 512/128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

Most Missing We can notice the first impact of the chunk selection strategy in figure 30 where the number of served clients with *Most Missing* has completely a different scaling tendency. This figure shows that around 2000

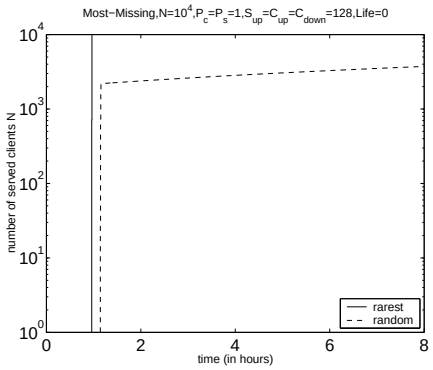
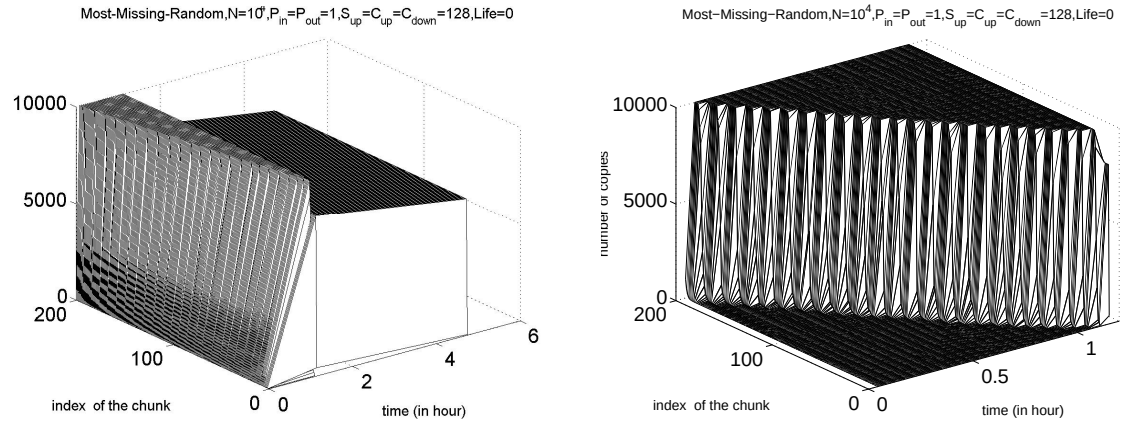


Fig. 30. The number of completed clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

clients finish at almost the same time, within **4160 seconds** of simulation. Then, the number of completed clients increases slowly. Indeed, it increases by one client each $\frac{1}{|C|}$ unit of time, which is equivalent to 16 seconds under the parameter values of table 9. To explain this behavior of *Most Missing Random*, we graph the chunks distribution over time in figure 31. If we take a closer look at this figure, we can observe that, after **4160 seconds** of simulation, all chunks are widely distributed in the system except one single chunk. In other words, all clients in the system have each **199 chunks** and they are all waiting for one rarest chunk to be scheduled from the server. Let us denote this rarest chunk by C_r. At time $t = 4160$ seconds, the server schedules chunk C_r to client 1. After 16 seconds, client 1 receives completely chunk C_r. Given that *Life* $= 0$, by receiving chunk C_r, client 1 completes its set of chunks and disconnects straight away. The server then delivers again this chunk C_r to a second client 2 and so on.



(a) Overall seen (b) Zoom over the first hour

Fig. 31. The chunks distribution over time for *Most Missing Random*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).

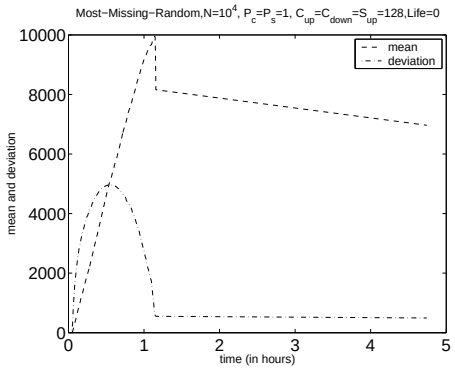


Fig. 32. The mean and deviation of the number of chunks against time for *Most Missing Random*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).

As a result, one client completes each 16 seconds and, instead of **3472 seconds** to serve 10^4 clients as with the rarest selection case, during 8 hours, *Most Missing Random* serves no more than **3733 clients**.

To confirm our analysis, we show in figure 33 a snapshot of the download progress of clients after 75 min (i.e. 4500 seconds) of simulations. After 75 min, around 20% of clients have completed their download and 80% are waiting for chunk C_r.

We give the following simplified scenario (figure 34) that helps to understand better why random selection of chunks can really block the clients in the system. Consider the case where there are only $N = 7$ clients that want to download a file that comprises only two chunks, C_1 and C_2. At time $t = 0$, the server starts serving chunk C_1 to client 1. After $\frac{1}{|C|}$ unit of time, the chunk is completely delivered and the server starts serving a new client 2. Given that the chunk selection is done at random, it is possible that the server schedules to client 2 the same chunk C_1 and not a new one. Meanwhile, client 1 uploads its chunk C_1 to a new client 3. At time $\frac{2}{|C|}$, the system includes 3 clients that hold chunk C_1 and four clients with no chunks at all. By that time, the server starts serving a new chunk C_2 to a new client 4. Similarly, clients 1, 2 and 3 upload their chunks to 3 new clients 5, 6, and 7. As a result, we land up with 6 clients that maintain chunk C_1 and one single client with chunk C_2. At time $\frac{3}{|C|}$, there are no new comers and existing clients can start exchanging their chunks. For sake of simplicity, assume that clients 1 and 4

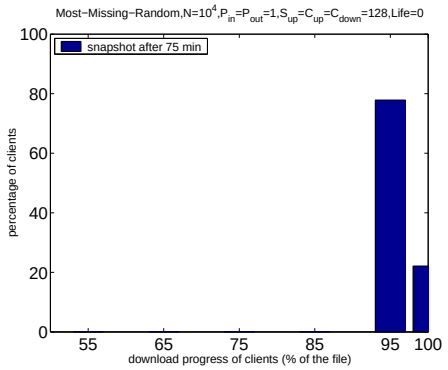


Fig. 33. The download progress of clients against time for *Most Missing Random*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$.

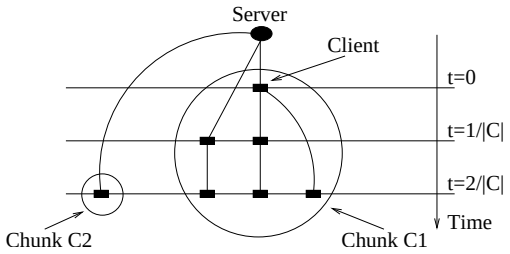


Fig. 34. The distribution of the chunks over the different clients during the first $\frac{2}{|\mathcal{C}|}$ unit of time in *Most Missing Random*. We assume that the number of clients is $N = 7$ and the number of chunks is $|\mathcal{C}| = 2$.

exchange their chunks, C_1 and C_2 respectively. At the same time, the server serves chunk C_2 to a client, say client 2. Remaining clients, 3, 5, 6, and 7, can not cooperate because they hold all the same chunk C_1 and thus remain idle. By time $t = \frac{4}{|\mathcal{C}|}$, clients 1, 2, and 3, complete their set of chunks and disconnect (*Life* $= 0$). As a result, at time $t = \frac{4}{|\mathcal{C}|}$, the system would comprise four clients (3,5,6, and 7) that are all waiting for the same chunk C_2 and each $\frac{1}{|\mathcal{C}|}$ unit of time, the server delivers that chunk to one client.

Least Missing In a linear chain architecture like *Least Missing*, the chunk selection strategy should not have impact on the performance of the system. The reason is that, the server keeps on serving the root of the chain, which in turn serves the next client and so on. Thus, each peer serves its chunks, one after the other to its successor in the chain. However, as already stated, the lack of synchronization between peers prevents *Least Missing* to behave as a perfect one and therefore, we expect the impact of the chunk selection strategy on *Least Missing* to be pronounced. As we can point out from figure 35, choosing the appropriate chunk to be scheduled is also a key performance for the *Least Missing* policy. When selecting rarest chunks first, *Least Missing* serves 10^4 clients within around 7 hours. In contrast, within 8 hours, *Least Missing* with random selection of chunks does not serve more than **1598 clients**. Thus, simplifying the system would give a very bad performance. To further confirm our analysis, we give the chunks distribution in figure 36(a). From figure 36(a) we can observe that the number of chunks increases for all chunks except one, which corresponds to the line on the right of the figure, which is the rarest chunk. Furthermore, from the simulation results (figure 36(b)), we know that at time $t = 19204$ seconds, there are **1791200 chunks** distributed over **9000 clients**, which makes 199 chunks per client. This means that, at time $t = 19204$ seconds, all clients miss the same chunk C_r and are waiting for it to be scheduled by the server.

These 9000 clients "stuck" in the system can be seen better through the snapshot of the download progress of clients after 5.5 hours (9800 seconds) of simulationsin (figure 37). This figure is after 5.5 hours (9800 seconds) of

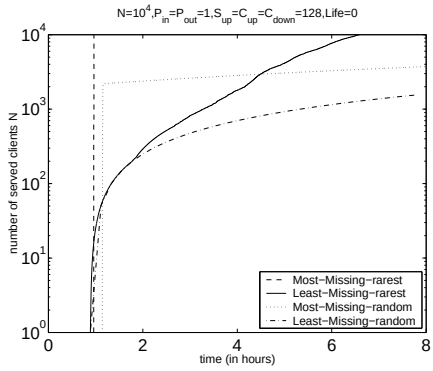
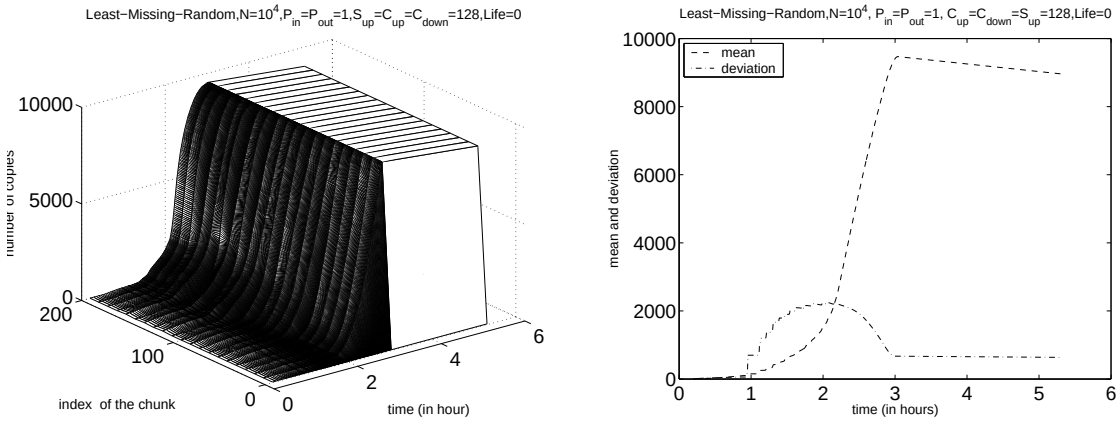


Fig. 35. The number of completed clients against time. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).



(a) Number of chunks vs time and chunk index

(b) Mean and deviation vs time

Fig. 36. The chunks distribution over time for *Least Missing Random*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).

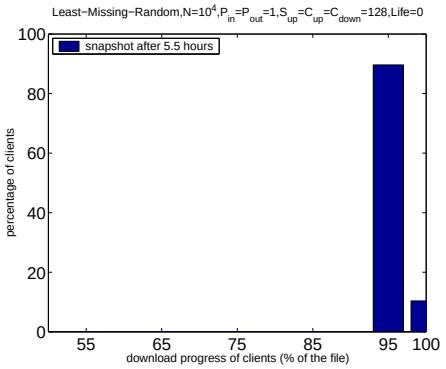


Fig. 37. The download progress of clients against time for *Least Missing Random*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).

simulations.

Conclusions As we stated at the beginning of this report, the chunk selection strategy is a main factor that draws the performance of the system. In this section we evaluated *Least Missing* and *Most Missing*, under the assumptions that peers schedule chunks at random and not rarest ones first. Our results showed a similar behavior in both architectures: A large fraction of clients are stuck in the system because they are all waiting for one rarest chunk to be served by the server.

6.6 Conclusions and Outlook

Conclusions We evaluated the performance of the *Most Missing* and *Least Missing* architectures under various assumptions on the system parameters. We started with a simple and basic scenario where we assumed (i) $S_{up} = C_{up} = C_{down} = 128$ Kbps, (ii) *Life* $= 0$ (clients are selfish), and (iii) $P_{in} = P_{out} = 1$. This scenario provided basic insights into how the cooperation between peers influences the performance of the system. We saw that *Most Missing* minimizes the overall service time as it engages clients into the delivery process very rapidly and keeps them busy during all their stay in the network. In contrast, *Least Missing* achieves a similar behavior to a linear chain; it serves quickly the first few clients in the system while the last client to be served experiences a high service delay. We also looked at the influence of different factors on the behavior of the two architectures. To summarize:

- *The indegree/outdegree of peers:* In a homogeneous environment and unconstrained bandwidth network, parallel upload and download of the chunks is not of benefit for *Most Missing* while it improves greatly the efficiency of *Least Missing*; as the indegree and outdegree of clients increase from 1 to 5, the time to serve 10^4 clients with *Least Missing* drops by 50%.
- *The life time of clients: Most Missing* minimizes the impact of the life time of clients as it holds them in the system as long as possible. In contrast, having altruistic clients is essential for *Least Missing* to achieve good results. When clients stay for 5 additional min in the system, the performance of *Least Missing* doubles.
- *The bandwidth heterogeneity of clients:* The main challenge here is how to prevent clients with low upload capacity from damaging the performance of the system. We have seen that *Least Missing* performs very badly under these conditions and clients may stay idle for too long before they receive new chunks. In contrast, by occupying all clients all the time, *Most Missing* ensures a high service capacity of the system and allows the clients to progress fast.
- *The chunk selection strategy:* Giving high priority to rarest chunks is a key design to ensure efficiency. When we allow peers to serve chunks randomly, clients get stuck in the system because they all miss the same rarest chunk and wait for the server to schedule it.

Outlook In the results that we have presented so far, we have assumed that all N clients access the system at the same time. We now continue our evaluation of the two architectures and consider the case where clients arrive to the system according to a Poisson process with rate λ. As before, we start with a basic and homogeneous scenario. In this scenario we highlight the influence of the arrival process on the behavior of the two architectures. We then validate our conclusions through more complex scenarios.

7 Simulation Results: Poisson Arrival of Clients

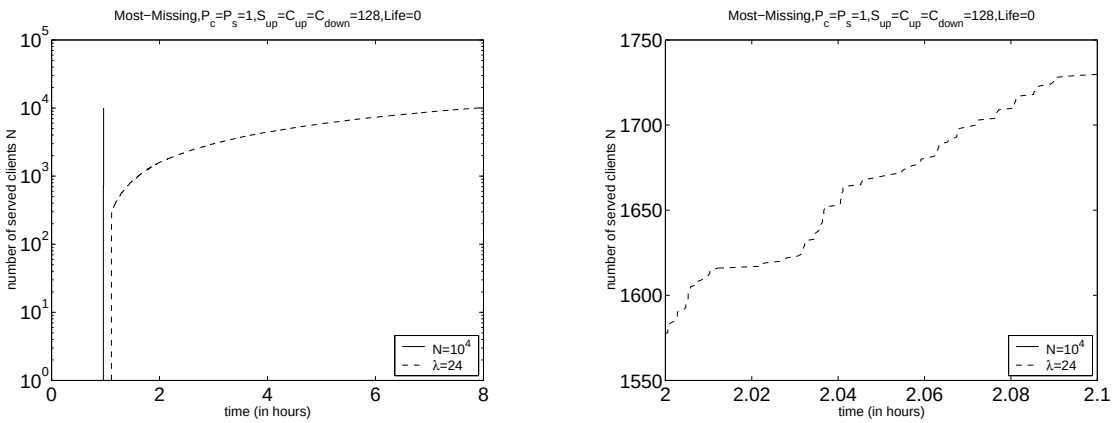
7.1 Basic Results

We now extend our analysis and study the scenario of continuous arrival of clients. We assume that clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. The parameter values that we consider in this basic and homogeneous scenario are presented in table 10.

Most Missing One would expect clients in *Most Missing* never to complete their download under a continuous arrival of clients: Existing clients will be always serving new ones and will never progress. Yet, this intuition is not correct. As we explained in the case of instantaneous arrivals, there is a start-up period where existing clients serve always new comers. But as soon as the number of new arrivals becomes less than the number of existing clients,

Table 10. Parameter values.

Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	$Life$
$\lambda = 24$ clients/min	128 Kbps	128 Kbps	128 Kbps	1	1	0



(a) Overall view (b) Zoom over a sub-set of clients

Fig. 38. The number of completed clients against time for *Most Missing*. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

existing clients start exchanging chunks amongst them and their download progress. Still, it would be interesting to figure out whether continuous arrivals delay the download progress of clients. For this purpose, we plot in figure 38(a) the number of served clients versus time. This figure shows that *Most Missing* is also very efficient under a continuous arrival of clients. As stated before, with the parameter values presented in table 10, in best cases, a client takes 3200 seconds to retrieve the whole file. Given that the simulation has lasted for 8 hours, only clients that arrive to the system 3200 seconds before the end of the simulation can be served. As a result, during 8 hours of simulation and with an arrival rate of $\lambda = 24$ clients/min, at most, $\frac{8 \cdot 3600 - 3200}{\frac{24}{60}} = 10240$ clients can be served. During these 8 hours, *Most Missing* has served **10153 clients** out of 10240 clients, i.e. more than **99**% of clients have completed their download. From figure 38 we can also make out that the number of served clients jumps suddenly to around 300 and it then increases progressively. To understand the reason, we zoom in figure 38(b) over a sub-set of clients. From this figure we can conclude a very important feature: *Most Missing* batches clients that arrive close in time and serves them together. In addition, except for clients that arrive at the beginning, the batches are not too long and consequently, this strategy does not really delay the download progress of clients. In fact, figure 39 shows that the residence time for the majority of clients is close to 3200 seconds, which is the optimal download time of the file[4]. The reason why the clients that arrive first in the system stay longer than subsequent ones can be explained as follows. At the beginning, the system has few clients and few chunks and consequently, a small upload capacity. Thus, a client may not find always servants for the chunks it needs. As time goes by, the upload capacity of the system and thus its potential service increases and clients can find more servants and can then progress faster. The above mentioned batches can also be seen from figure 40 where we give a snapshot of the download progress of clients after 4 hours of simulations. This figure shows how clients in progress are distributed over different batches, with each batch comprises similar number of clients.

As concerns the chunks evolution over time, figure 41(a) reveals once again that the chunks propagate in the system at a high speed. As compared to an instantaneous arrival of clients, we notice a new behavior here: Chunks that are injected later catch faster with earlier ones and consequently, the deviation drops faster to zero (figure

[4] We define the residence time of a client as the time elapsed between the moment the client joins the network and the moment it leaves the network.

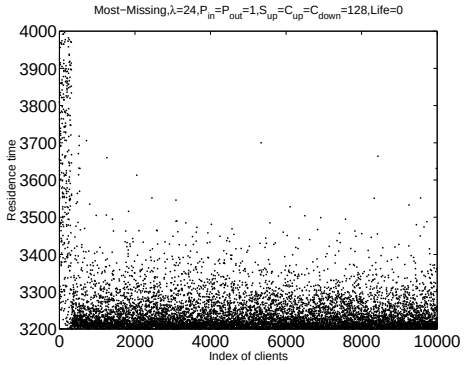
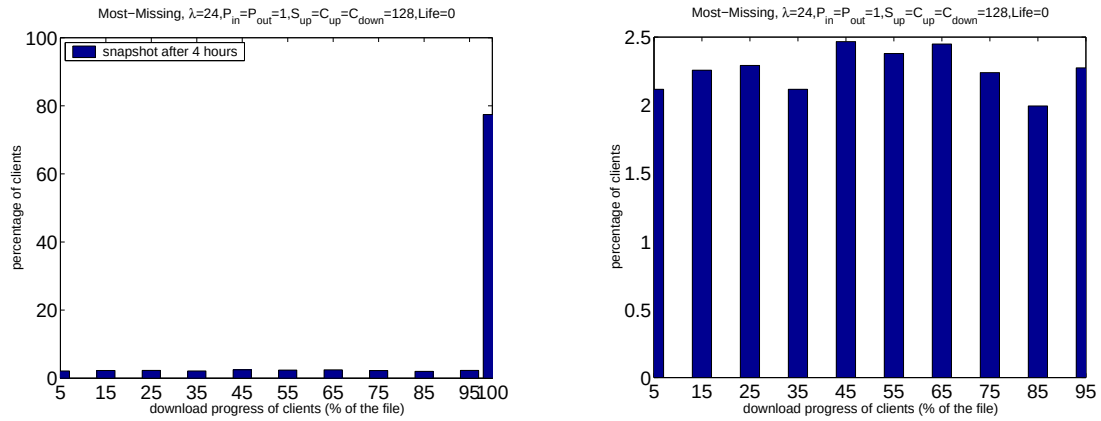


Fig. 39. The residence time of clients for *Most Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).



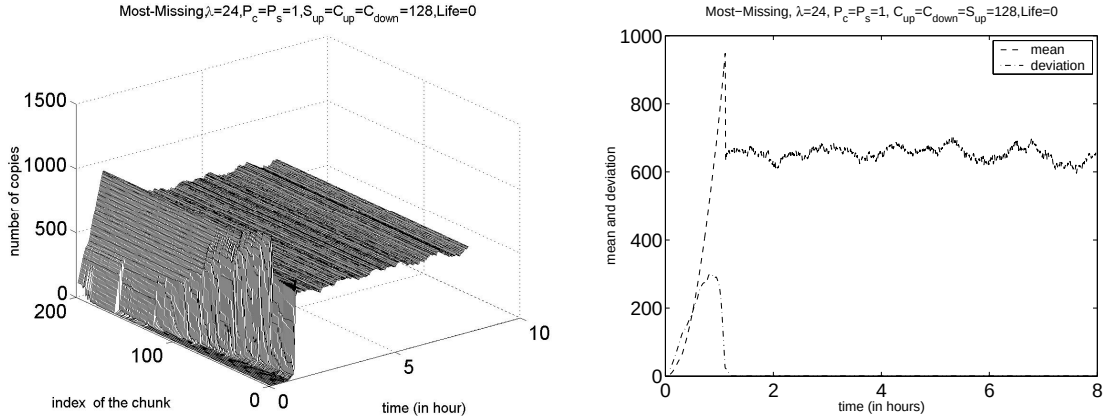
(a) overall view (b) zoom over non completed clients

Fig. 40. Snapshot of the download progress of clients for *Most Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

41(b)). This behavior is logic because, under a Poisson arrival with rate $\lambda = 24$ clients/min, the start-up period[5] is shorter and the exponential expansion of the number of chunks lasts for a shorter time.

Least Missing We now evaluate the *Least Missing* policy. What might seem surprising here is that, in case of a Poisson arrival of clients, *Least Missing* and *Most Missing* have almost the same performance (figure 42). Within 8 hours of simulation, *Least Missing* serves **10163 clients** and *Most Missing* serves **10153 clients**. Under the parameter values of table 10, a perfect *Least Missing* (i.e. a single linear chain) serves only **1600 clients** within 8 hours. This result shows again that the behavior of *Least Missing* is not for a perfect one and it is highly dependent on the scenario itself. What is also interesting in figure 42 is that, at the beginning, the number of served clients follows the curve for *Least Missing* under an instantaneous arrival of clients. It then increases suddenly at around $t = 2.5$ hours and catches up with the curve of the *Most Missing*. In fact, due to the lack of synchronization between peers, this strategy delivers few chunks to clients that are not least missing. As time goes by, these non

[5] Recall that the start-up period is the initial phase where the number of new comers is larger than the number of existing clients in the system.



(a) Number of chunks vs time and chunk index (b) Mean and deviation vs time

Fig. 41. The chunk distribution over time for *Most Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

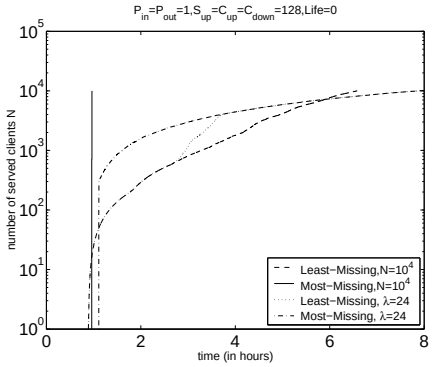


Fig. 42. The number of completed clients against time. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

least missing clients become an important potential service and allow new comers to get the chunks faster. This explains why clients that join the system a bit late notice less download time (figure 43).

The improvement in the performance of *Least Missing* can also be seen if we observe the chunk distribution over time in figure 44. As compared to an instantaneous arrival of clients (figure 14), figure 44 shows that the number of copies of the different chunks increases faster.

Conclusions In this section we investigated the influence of a continuous arrival of clients on the system behavior. We assumed that clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. The results that we obtained are similar to those for the scenario with instantaneous arrivals. *Most Missing* optimizes always the average service time over all clients. To achieve such a service, *Most Missing* forces clients that arrive first to the system to stay for a bit longer than those that arrive later on. We can imagine that, by doing so, the system tries to capitalize an initial service capacity.

The *Least Missing* strategy, on the other hand, optimizes as before the service time of the first few clients. What is new here is that the performance of *Least Missing* is much better than before. The reason is that, due to the lack of synchronization, the system serves many clients that are not least missing and with time, these non least missing clients increase the service capacity of the system, which benefits clients that arrive afterwards.

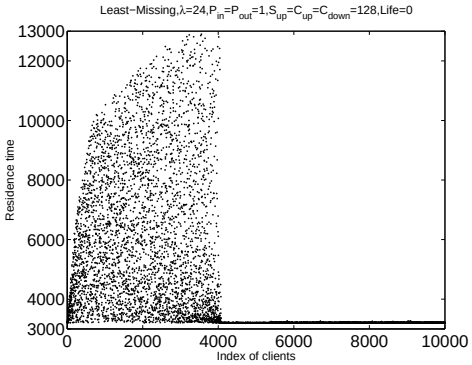
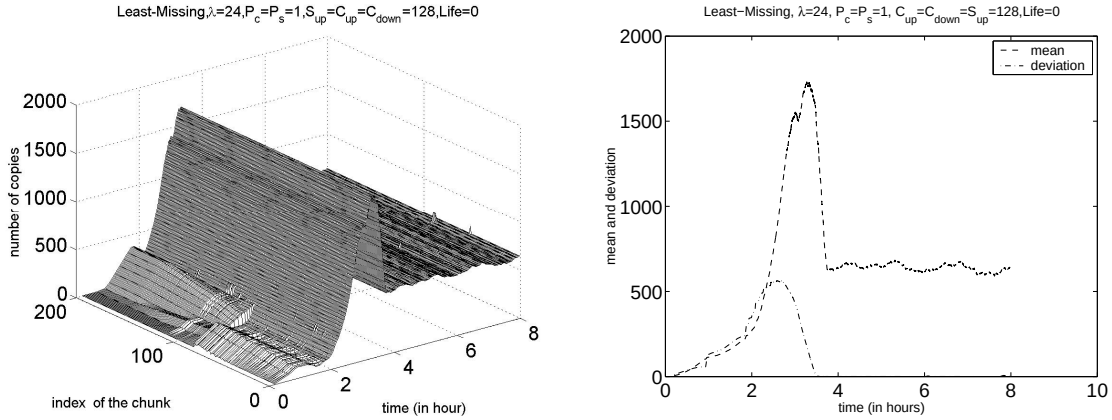


Fig. 43. The residence time of clients for *Least Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).



(a) Number of chunks vs time and chunk index (b) Mean and deviation vs time

Fig. 44. The chunks distribution over time for *Least Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

7.2 Impact of the Indegree/Outdegree of Peers

For instantaneous arrival of clients, we have seen that, parallel download and upload of the chunks is not of benefit to *Most Missing* while it improves greatly the performance of *Least Missing*. The simulations that we conducted under a Poisson arrival of clients (with the parameter values as depicted in table 11) have exhibited the same tendency (figure 45).

Table 11. Parameter values.

Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
$\lambda = 24$ clients/min	128 Kbps	128 Kbps	128 Kbps	5	5	0

The only difference is that, in case of *Least Missing*, the two curves for ($P_{in} = P_{out} = 1$) and ($P_{in} = P_{out} = 5$) are identical after 4 hours. From section 6.2 we know that large indegrees/outdegrees allow *Least Missing* to engage

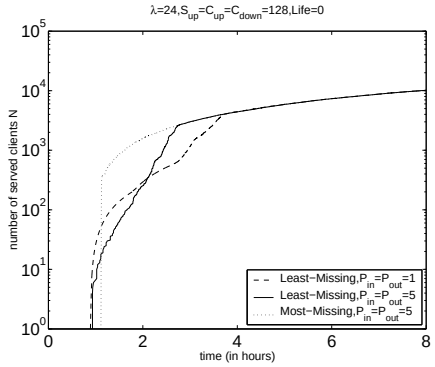


Fig. 45. The number of completed clients against time. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

clients fast into the file delivery, which improves the system performance. But from section 7.1 we also know that, under Poisson arrivals, clients that arrive late to the system receive a very good service. Thus, this improvement due to large indegrees/outdegrees concern only clients that arrive early to the system. This explains why after 4 hours of simulations, we notice no change between *Least Missing* with $P_{in} = P_{out} = 1$ and *Least Missing* with $P_{in} = P_{out} = 5$. Figure 46 summarizes what we just explained. The residence time of clients within the first few

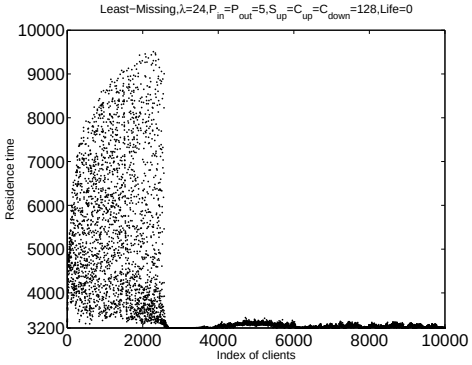


Fig. 46. The residence time of clients for *Least Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most 5 download and 5 upload connections ($P_{in} = P_{out} = 5$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

hours decreased while it did change for subsequent clients.

7.3 Impact of the Life Time of Clients

The life time of clients is not an important parameter for *Most Missing* under the scenario where all peers arrive to the system at the same time. The reason is that all clients finish at almost the same time and no need to have volunteer clients because there remains no clients to be served. However, when clients arrive to the system progressively, as under a Poisson arrival, the delivery process continues over many client generations. Thus, one would expect that having additional sources for the full file helps new comers to progress fast. Yet, *Most Missing* proves once again that it does not need altruistic clients. In figure 47 we plot the number of served clients versus time for both *Most Missing* and *Least Missing* approaches. The parameter values used here are showed in table 12.

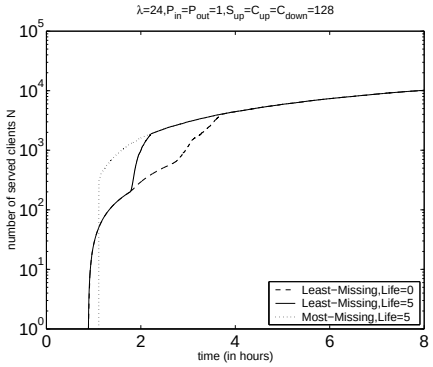


Fig. 47. The number of completed clients against time. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps.

Table 12. Parameter values.

Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
$\lambda = 24$ clients/min	128 Kbps	128 Kbps	128 Kbps	1	1	5

As compared to figure 38(a), figure 47 shows clearly that forcing clients to stay for 5 additional min in the system to help other clients is not necessary for *Most Missing*. This can be the case only when all existing clients in the system are fully using their download and upload capacities during all their residence time in the system.

On the other hand, figure 47 confirms the previous results: The performance of *Least Missing* increases with the life time of clients. However, as for $P_{in} = P_{out} = 5$, this improvement in the performance is just for clients that arrive early to the system. After 4 hours of simulations, there already exist a lot of clients in the system that can serve new comers and therefore, the advantage of *Life* $= 5$ is discounted.

7.4 Impact of the Bandwidth Heterogeneity of Clients

Given that *Least Missing* has basically similar tendency to a linear chain, the performance of the system can be highly damaged by the clients with the lowest bandwidth capacity. This result was the case under an instantaneous arrival of clients and we show here that it holds true under continuous arrivals. Figure 48 demonstrates that *Least Missing* performs very badly in heterogeneous environments. Table 13 points out the values that we consider for the different parameters in this section.

Table 13. Parameter values.

Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
$\lambda = 24$ clients/min	128/64 Kbps	512/128 Kbps	1024 Kbps	1	1	0

In contrast to *Least Missing*, *Most Missing* leverages more efficiently the clients heterogeneity. From figure 48 we can see that the overall performance of the system is quite close to the homogeneous scenario. The main difference is that the time at which the first batch of clients is served has doubled. In other words, the system increases the initial phase where it tries to capitalize an important service capacity. However, once the number of active clients in the system (i.e. the service capacity) becomes quite high, the initial batch is served and next clients experience similar performance to the homogeneous scenario. As a result, only clients that arrive at the beginning notice a high download time while the remaining clients receive a very good service.

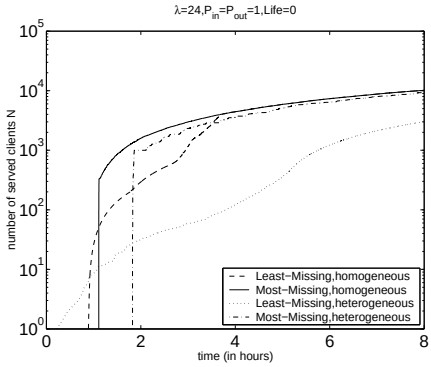


Fig. 48. The number of completed clients against time. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). Clients disconnect as soon as they complete their download (*Life* $= 0$).

7.5 Impact of the Chunk Selection Strategy: Random rather than Rarest

We now evaluate the two strategies *Most Missing Random* and *Least Missing Random* under a Poisson arrival of clients for the parameter values outlined in table 14. The scaling behavior of the two architectures (figure 49)

Table 14. Parameter values.

Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
$\lambda = 24$ clients/min	128 Kbps	128 Kbps	128 Kbps	1	1	0

confirm once again the importance of the chunk selection strategy as a key design for the system.

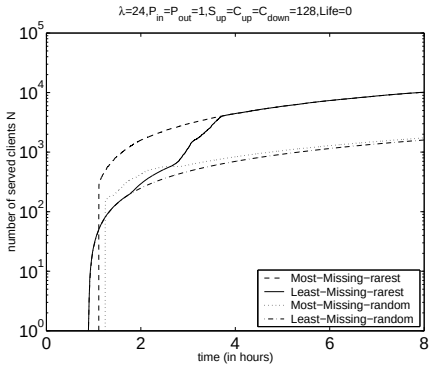


Fig. 49. The number of completed clients against time. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

7.6 Conclusions

In this section we addressed the impact of the arrival process of clients on the system behavior. To this purpose, we assessed the performance of *Most Missing* and *Least Missing* when clients arrive according to a Poisson process with rate $\lambda = 24$ clients/min. As a summary, we have seen that *Most Missing* tends always to optimize the average service time seen by clients while *Least Missing* provides a good service for the clients that arrive first to the system. In addition, the broad conclusions that we drew concerning the impact of the different parameters on the two architectures remain valid. For instance, the indegree/outdegree of peers is key for the performance of *Least Missing* while *Most Missing* is very insensitive to this parameter. Similarly for the life time of clients. On the other hand, *Most Missing* handles better than *Least Missing* the bandwidth heterogeneity of clients and finally, the selection strategy is an essential feature for both approaches.

8 Open Questions

We have seen so far how the performance of the system is affected by the way clients organize and cooperate. From the scenarios that we considered, we can easily conclude that there is no "optimal strategy" as each strategy provides various trade-offs and may prove adequate for specific goals and deployment scenarios. The two strategies that we considered in this report represent two extreme ways for cooperation between peers. However, the importance of the analysis that we gave is two fold. First, it highlights the main factors that influence the performance of the system. Second, it provides new insights into where to apply what rule, which helps in the design of new cooperative architectures depending on the goal and the environment conditions. Moreover, these two strategies pave the way to conclude many of other strategies as a combination. One combination is the *Most Adaptive* strategy proposed in [5]: Clients that have many chunks serve clients that have few chunks, and vice-versa, with more randomness introduced when download tend to be half complete.

This strategy gives good chances to new comers without artificially slowing down clients that are almost complete.

In our analysis, we mainly carried on the performance aspect of mesh approaches. Yet, there are still many important aspects that must be addressed:

- P2P architectures today suffer from free riders, i.e. clients that access the service and give nothing in return. One interesting question would be to prevent such a practice.
- Another challenging question is whether we could ensure efficiency and fairness at the same time. By fairness we mean that clients that help the most in the file delivery should receive the best service.
- The two architectures that we studied include no optimizations at all. One optimization would be to serve first the clients that provide a high upload capacity, which would help the system to engage faster clients into the file delivery and improves its performance. Yet, this kind of optimizations includes the challenge of how to find the clients with the largest upload capacities. One can use similar solution to the one proposed by *Slurpie* [10], where each client reports to the server the information about the connection it is using.

9 Conclusions and Future Work

9.1 Summary

File replication in P2P networks is becoming an important service in the current Internet. Existing approaches for file replication can be largely classified into tree and mesh approaches. Tree approaches can be very efficient and scale exponentially in time. However, we believe that constructing and maintaining the trees in a dynamic environment like P2P networks is a very challenging problem. In contrast to tree approaches, mesh approaches are very flexible and more robust against bandwidth fluctuations and client departures.

In this report we achieved two main goals. The first one was to prove that mesh approaches can be at least as efficient as tree approaches. To do so, we introduced two new cooperation architectures, namely *Least Missing* and *Most Missing*. Each architecture includes a peer selection strategy coupled with a chunk selection strategy. The first architecture, *Least Missing*, favors clients that have many chunks and the second one gives more priority to clients that have few chunks. In both cases, rarest chunks are scheduled first. We analytically proved that *Least Missing*

can simulate a tree distribution with an outdegree k in a simpler way. The key idea is to set the indegree of peers to $P_{in} = 1$ and the outdegree to $P_{out} = k$. We also showed via simulations that *Most Missing* achieves similar performance of $PTree^k$ while avoiding the overhead of constructing multiple trees. We outline that, along this report, we assumed a network with no bandwidth constraints and the only constraint is the upload and download capacity of peers. The rational behind such an assumption is that we wanted to focus on the advantages and shortcomings related to our architectures and not to external factors. We also assumed that each client in *Least Missing* and *Most Missing* knows all other clients in the system.

The second goal of this report was to perform a complete analysis that provides guidelines for the design of new architectures. We started our analysis with a basic scenario where we assumed that:

- All peers (server and clients) have equal upload and download capacities, $S_{up} = C_{up} = C_{down} = r$.
- Each client stays online until it receives the whole file.
- Peers have equal outdegree and indegree $P_{out} = P_{in} = 1$.
- All N clients arrive to the system at time $t = 0$.

The condition of instantaneous arrival of clients represents a crucial scenario that may be the case where (i) A critical data, e.g. anti-virus, must be updated over a set of machines as fast as possible or (i) A flash crowd, i.e. a large number of clients that arrive to the system very close in time. Our basic scenario provided new insights into the basic tendency of each of the two architectures. The impact of the peer selection strategy on the system performance was clear: *Most Missing* optimizes the average download time overall clients while *Least Missing* provides a very good service to a few clients at the expense of a larger download time for the remaining ones.

We then isolated the main parameters that can influence the system performance. Our conclusions are:

- The indegree/outdegree of peers can have a strong to little effect on the scaling behavior of the system. As P_{in} and P_{out} go from 1 to 5, the performance of *Most Missing* slightly degrades while the performance of *Least Missing* doubles. Even though, we argue that parallel download and upload of the chunks can offer many advantages in real environments. Mainly, it allows clients to fully use their upload and download capacities, which makes the system more robust against bandwidth fluctuations in the network and client departures. Also, parallel connections achieve a good connectivity between peers in the system.
- The behavior of clients can not be predicted in advance, some clients disconnect straight after they receive the file and some others stay and help into the file delivery. In this report we evaluated the gain in performance in case of altruistic clients that stay in the networks for 5 additional min after they complete their download. Our results showed that this behavior is suitable for architectures that serve clients sequentially like *Least Missing*. In contrast, for strategies like *Most Missing* where clients stay as long as possible and finish at almost the same time, this parameter has no effect.
- The bandwidth heterogeneity of clients can dramatically affect the performance of the system especially when clients are served sequentially, which is the basic tendency of *Least Missing*. In such an architecture, clients may wait for too long to receive new chunks. *Most Missing*, on the other hand, minimizes the impact of the clients heterogeneity. By keeping all peers busy most of the time, clients can always find servants to download the chunks they miss and could progress fast.
- The chunk selection process is a key design to ensure efficiency. Both architectures, *Most Missing Random* and *Least Missing Random* have resulted in a very low performance: A large fraction of clients are stuck in the system because they all miss the same rarest chunk and are waiting to receive it from the server.
- We validated our results under both, instantaneous and Poisson arrival of clients. What is new under a Poisson arrival is that *Most Missing* forces clients that arrive first to the system to stay for a longer time than subsequent ones. The reason is that, at the beginning, the system comprises few clients with few chunks and has a limited upload capacity. Thus, clients do not always find servants for the chunks they miss. As time goes by, the system incorporates more clients and its potential service grows significantly, which benefits clients that arrive later on. This same result applies also to *Least Missing*.

Our analysis and conclusions highlight the main parameters the system designer must take into account. In addition, the range of scenarios that we considered helps the design of an efficient cooperative architecture depending on the goals to achieve and the environment conditions. Note that our results are not only limited under instantaneous and Poisson arrival of clients, but they also apply when the arrival process is bursty. Actually, the more bursty the arrivals, the closer we become to the instantaneous scenario. In contrast, the less bursty, the closet we are to the Poisson scenario.

9.2 Future Work

In the above discussion, we made some assumptions in order to simplify the analysis. As a future work, one could look at the following scenarios:

- The results that we gave so far are for a static network where we assumed no failures and that the bandwidth over each link is equally divided amongst the different connections which share that link. One interesting extension would be to evaluate the two architectures, *Most Missing* and *Least Missing*, in real network. The goal behind this extension is to figure out how far the network characteristics prevent the system to meet its goals.
- In our simulator, we assumed that each client has a perfect knowledge of the network. One natural extension would be to limit the number of neighbors a client can communicate with, i.e. consider a local view of the system. We do not expect such an assumption to change dramatically the tendency of the two architectures. We believe that a small list of neighbors, $40 - 120$ per client as in *BitTorrent* [4], is enough to perform a cooperation strategy between clients.
- Our analysis assumed no early departure of clients, which is not the case in practice. Clients in P2P networks can fail or disconnect at any moment. For the *Least Missing* strategy, this factor has no impact because clients are served in a chain; the parent of a failed client automatically reconnects to the next client in the chain. In *Most Missing* the scenario is a bit different. When a client disconnects, the system looses a potential server but at the same time, this failed client would not require any further chunks from the system. Therefore, the overall upload to download capacity of the system would not change significantly.
- In all scenarios, we assumed that clients either disconnect straight after the reception of the file or stay for 5 additional min. In the evaluation that we performed for *BitTorrent* [8], we learned that the distribution of the life time of clients can vary significantly, from 2 min to 6 hours. Thus, one extension would be to evaluate the two architectures for a similar distribution.

References

1. "BitTorrent beats Kazaa, accounts for 53% of P2P traffic", AlwaysOn Site, July 2004, http://www.alwayson-network.com/index.php.
2. E. W. Biersack, P. Rodriguez, and P. A. Felber, "Performance Analysis of Peer-to-Peer Networks for File Distribution", In *Proc. of QofIS*, Barcelona, Spain, September 2004.
3. M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth content distribution in a cooperative environment", In *Proc. of IPTPS*, Berkeley, CA, USA, February 2003.
4. B. Cohen, "Incentives to Build Robustness in BitTorrent", In *Proc. of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003, http://bitconjurer.org/BitTorrent.
5. P. A. Felber and E. W. Biersack, "Self-scaling Networks for Content Distribution", In *Proc. of Self-**, Bertinoro, Italy, May 2004.
6. C. Gkantsidis, M. Ammar, and E. Zegura, "On the Effect of Large-Scale Deployment of Parallel Downloading", In *Proc. of WIAPP*, San Jose, CA, June 2003.
7. B. Hill, "P2P: 70-80 Percent of All EuroNet Traffic", The Digital Music Weblog, May 2004, http://digitalmusic.weblogsinc.com/.
8. M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. A. Hamra, and L. Garcés-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime", In *Proc. of PAM*, Juan-les-Pins, France, April 2004.
9. L. E. Schrage, "A Proof of the Optimality of the Shortest Remaining Service Time Discipline", *Operations Research*, 16:670–690, 1968.
10. R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol", In *Proc. of INFOCOM*, Hong-Kong, China, March 2004.
11. X. Yang and G. de Veciana, "Service Capacity of Peer-to-Peer Networks", In *Proc. of INFOCOM*, Hong-Kong, China, March 2004.