Design Choices for Content Distribution in P2P Networks*

Anwar Al Hamra
IUT of Nice
GTR Departement
Sophia Antipolis, FRANCE
alhamra@iutsoph.unice.fr

Pascal A. Felber
Université de Neuchâtel
Institut d'informatique
Switzerland
pascal.felber@unine.ch

ABSTRACT

Content distribution using the P2P paradigm has become one of the most dominant services in the Internet today. Most of the research effort in this area focuses on developing new distribution architectures. However, little work has gone into identifying the principle design choices that draw the behavior of the system. In this paper, we identify these design choices and show how they influence the performance of different P2P architectures. For example, we discuss how clients should organize and cooperate in the network. We believe that our findings can serve as guidelines in the design of efficient future architectures.

Categories and Subject Descriptors

C.2.0 [**General**]: Data communications; C.2.1 [**Network Architecture and Design**]: Network communications; C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Design, Performance

Keywords

peer to peer networks, content distribution, performance evaluation

1. INTRODUCTION

Peer-to-peer systems are distributed Internet applications in which the resources of a large number of autonomous participants are harnessed in order to carry out the system's function. In such systems, peers connect to each others and form an application overlay network on top of conventional Internet protocols. P2P systems are experiencing a great success mainly because they are self-organized, very reliable, and fault tolerant.

The P2P paradigm has been initially discussed for file sharing services such as Napster and Gnutella. However, P2P networks represent also an efficient infrastructure for content distribution. The most promising property of P2P distribution networks is that the responsibility of distributing the content is spread amongst downloaders, which greatly reduces the load on the server. In these networks, clients contribute to the resources of the system as a function of their upload capacities. Clients download the content and upload it to other clients and so on. As a consequence, P2P distribution networks are seen as an ultimate solution to scalability. The larger the number of clients in the system, the larger its service capacity and the better it scales and serves the content.

There are two popular solutions to distribute the content in P2P networks. The first one organizes clients in a mesh. In mesh-based approaches, each node knows a subset of clients, called neighbors. Neighbors cooperate and exchange the content they hold according to a predefined "cooperation strategy". The second solution is to construct a distribution tree on top of clients. Tree-based approaches can be used under a particular scenario where clients arrive to the system very close in time. Biersack et al. [1] recently showed that tree-based approaches can be very efficient for distributing a file to a large number of clients. The authors proved that the number of served clients in tree-based approaches scales exponentially in time.

Given these two solutions, we still do not know which one performs the best. Thus, a first design choice is to select the way clients must organize themselves in the network. In the first part of the paper, we compare the number of served clients over time in both, tree-based and mesh-based approaches. Our results demonstrate that, mesh-based approaches take less time than tree-based approaches to serve the same number of clients. To the best of our knowledge, this comparison has never been done previously.

Once we know how clients must be organized, i.e. in a mesh, the second design issue is related to the way peers[1] should cooperate to retrieve the content. We need clever cooperation strategies to efficiently leverage the available bandwidth resources at the different peers and to rapidly distribute the content. A cooperation strategy is the result of many factors coupled together. These factors include mainly (i) the peer selection strategy, (ii) the chunk selection strategy, and (iii) the network degree. Given a set of neighbors, the peer selection strategy indicates which neighbor must be served first. Once the neighbor to be served is chosen, the chunk selection strategy tells the peer which chunk of the content must be transmitted. We assume that the content is divided into pieces called chunks. Finally, the network degree represents the maximum number of active download and upload connections a peer can simultaneously maintain. In the second part of the paper, we investigate the impact of the above factors on the cooperation between peers. We show through examples and simulations how these factors play a key role in the behavior of mesh-based approaches.

*This work has been entirely done at Institut Eurecom

[1]We use the term peer only when we want to refer to both, clients and server at the same time.

Our results and discussions provide interesting hints and observations that can be very helpful in designing efficient distribution architectures.

The rest of the paper is structured as follows. In section 2 we introduce the parameters that we use in the sequel. Section 3 briefly overviews the basic design of tree-based approaches while section 4 discusses mesh-based ones. In section 5 we compare the time needed to serve the same number of clients in both, tree-based and mesh-based approaches. We then focus on mesh-based approaches and extensively investigate their performance in section 6. We conclude the paper in section 7 with a discussion of the results.

2. NOTATION

We denote by $\mathcal{C}$ and $|\mathcal{C}|$ respecively the set and number of all chunks in the file being distributed. $\mathcal{D}_i$ corresponds to the set of chunks that client i has already downloaded. In contrast, $\mathcal{M}_i$ represents the set of chunks that client i is still missing. Under these assumptions, we obtain $\mathcal{M}_i \cup \mathcal{D}_i = \mathcal{C}$ and $\mathcal{M}_i \cap \mathcal{D}_i = \emptyset$. Similarly, $d_i \triangleq \frac{|\mathcal{D}_i|}{|\mathcal{C}|}$ denotes the proportion of chunks that client i has already downloaded. $m_i \triangleq \frac{|\mathcal{M}_i|}{|\mathcal{C}|}$ corresponds to the proportion of chunks that client i is still missing.
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 P_{in} and the outdegree by P_{out}. *one unit of time* is the time needed to download the file at rate r, where r is a constant expressed in Kbps. It follows that, for a file of $|\mathcal{C}|$ chunks of equal size, $\frac{1}{|\mathcal{C}|}$ unit of time is needed to download one single chunk at rate r.
Finally, *Life* 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 immediately after it finishes the download.

3. TREE-BASED APPROACHES

As their name indicates, tree-based approaches construct a tree to distribute the content to clients. These approaches differ in the strategy employed to construct the tree. They also differ in the algorithms used to handle bandwidth degradation on a link or clients failure.
A recent paper by Biersack et al. [1] analyzes the performance of three tree-based architectures namely, *Linear*, $Tree^k$, and $PTree^k$. These architectures cover almost all existing tree-based ones. In their analysis, the authors assume that all N clients arrive to the system at time $t = 0$ and all clients have homogeneous and symmetric bandwidth capacities. They also assume that the server stays online indefinitely. In this section we briefly overview the basic idea of each of these architectures. This overview will facilitate the comparison that we perform next between tree-based and mesh-based approaches.

***Linear* Architecture.** *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. When the server uploads the entire file to client 1, it becomes free and starts serving a new client. As a consequence, a new chain is initiated and the same process above is repeated. Under these assumptions, the number of served clients with *Linear* is shown to increase quadratically in time.

Tree Distribution with Outdegree k ($Tree^k$). With $Tree^k$, clients are organized in a regular tree with an outdegree k. In $Tree^k$, an interior node in the tree serves the file to k clients simultaneously. Comparing to *Linear*, Biersack el al. demonstrate that this architecture performs much better because clients are served in parallel rather than sequentially. The number of served clients in $Tree^k$ increases exponentially in time. However, the main drawback of $Tree^k$ is that clients that are located at the leaves of the tree do not help in distributing the file.

Forest of Parallel Trees ($PTree^k$). This architecture is inspired from SplitStream [2] proposed initially for live streaming applications. With $PTree^k$, the file is split into k parts and each part is distributed over an independent tree rooted at the server. To construct these independent trees, $PTree^k$ requires each client to be an interior node in one tree and a leaf node in the remaining ones. This architecture performs the best compared to *Linear* and $Tree^k$. In $PTree^k$, all clients finish downloading the file rapidly and almost at the same time.
There are two interesting properties that make this architecture highly efficient. The first one is that clients are served in parallel as in $Tree^k$. The second property is that, by constructing parallel trees, this architecture ensures that all clients help in distributing the file.

4. MESH-BASED APPROACHES

Usually tree-based approaches require the server to run complex algorithms to construct and maintain the distribution trees. In contrast, in mesh-based approaches clients self-organize in a mesh. The way clients organize and cooperate is responsible of the performance of mesh-based approaches. A cooperation strategy between peers consists of a peer selection strategy coupled with a chunk selection strategy. We also add the network degree as a critical factor that influences the behavior of a cooperation strategy. Recall that the network degree is represented by the maximum number of active download and upload connections a peer can simultaneously maintain, i.e. P_{in} and P_{out}.

4.1 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 a large enough subset of other clients. For instance, a client in *BitTorrent* [3] has between 40 to 120 neighbors as we observed in [6].

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 paper, we consider 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 [7].

- *Most missing.* Preference is given to the clients that have few chunks (new comers), i.e., we serve in priority client j with $d_j \leq d_i$, $\forall\ i$. We expect this strategy to engage clients into the file delivery very fast and keep them busy for most of the time.

We note that *Least Missing* and *Most Missing* have been already introduced by Felber et al. in [4]. In this paper we use these two strategies to achieve two goals. The first one is to prove that mesh-based approaches are more efficient than tree-based ones. The second goal is to explain the importance of the peer selection strategy as a key design factor for mesh-based approaches.

4.2 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 chunk selection strategy aims at modifying the way the chunks get duplicated in the network. A good 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.

4.3 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 throughput. 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 distributed in the network. For instance, for a tree distribution, a small outdegree of $k = 2$ is optimal [1]. In our analysis, we will consider different values for the indegree P_{in} and outdegree P_{out} of peers.

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 paper 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 [2, 1] has advised to split the file into various "chunks", which permits concurrent downloads from multiple peers. In addition, with this technique, instead of waiting to download the whole file, a client can start serving a chunk as soon as it finishes downloading it. Still, 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 number and size of chunks based upon the required goal. Along this paper, we consider a common chunk size of 256 KB and a number of chunks $|\mathcal{C}| = 200$, which makes a file of 51.2 MB. We also consider the case of one single file.

5. MESH-BASED VS. TREE-BASED APPROACHES

In this section, we compare the number of served clients in mesh-based and tree-based approaches. First of all, we compare *Least Missing* to $Tree^k$. We demonstrate that *Least Missing* can achieve a similar performance to a tree distribution in a simpler way. Then we compare *Most Missing* to $PTree^k$. We show that *Most Missing* performs better than $PTree^k$ while avoiding the penalty of constructing parallel trees.

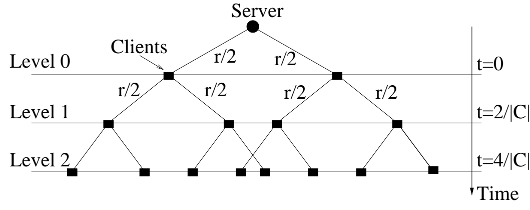
5.1 *Least Missing* vs. $Tree^k$

In *Least Missing*, each peer tries to serve first the neighbor that has the largest number of chunks amongst all other neighbors. Remind 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, we assume $k = 2$. We analytically prove that *Least Missing* with $P_{in} = 1$ and $P_{out} = 2$ is equivalent to a tree distribution with an outdegree $k = 2$. In this analysis, we make the following assumptions.

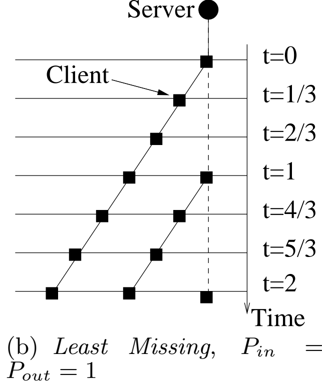
- All peers have equal upload and download capacity $S_{up} = C_{up} = C_{down} = r$ and maintain an outdegree $P_{out} = 2$ and an indegree $P_{in} = 1$.
- A client can start serving the file once it receives a first chunk.
- Each client remains online until it delivers twice the number of chunks it 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 $t = \frac{2}{|\mathcal{C}|}$. Recall that one unit of time is the time needed to download the whole file at rate r and that the file comprises $|\mathcal{C}|$ chunks of equal size. 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. At time $t = \frac{2}{|\mathcal{C}|}$, client 1 holds chunk C_1 and client 2 holds chunk C_2. 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. This same process repeats and one new level is added each $\frac{2}{|\mathcal{C}|}$ unit of time (see figure 1(a)). Level i includes 2^{i+1} clients, which are served by the 2^i clients located at level $(i-1)$. As a result, the number of clients

in the system evolves as in a tree with an outdegree $k = 2$. We can easily verify that the scenario $P_{in} = P_{out} = 1$ gives



(a) *Least Missing* with $P_{in} = 1$ and $P_{out} = 2$



(b) *Least Missing*, $P_{in} = P_{out} = 1$

Figure 1: Scaling behavior of *Least Missing* vs. time for different values of P_{in} and P_{out}. The black circle represents the server while the black squares represent the clients

a linear chain as described in figure 1(b). In this figure, we assume that the number of chunks is $|\mathcal{C}| = 3$. After one unit of time, the server finishes uploading the file to the head of the first chain. It then starts serving a new client and consequently, a new chain is initiated. More formally, a new chain starts each one unit of time until all clients are served.

5.2 *Most Missing* vs. $PTree^k$

Having seen how *Least Missing* can scale exponentially in 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 same properties that make $PTree^k$ very efficient. To validate our intuition, we compare the time needed to serve $N = 10^4$ clients with *Most Missing* and $PTree^k$ under the scenario given in table 1. Note that all $N = 10^4$ clients are assumed to arrive simultaneously at time $t = 0$ and that the server stays online indefinitely. In

Table 1: Parameter values

C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
128 Kbps	128 Kbps	128 Kbps	1	1	0

figure 2 we plot the number of served clients against time for *Most Missing* as obtained from simulations[2]. This figure proves that *Most Missing* behaves similarly to $PTree^k$. This means that all clients complete very fast and almost at

[2]Details about our simulator are given in section 6

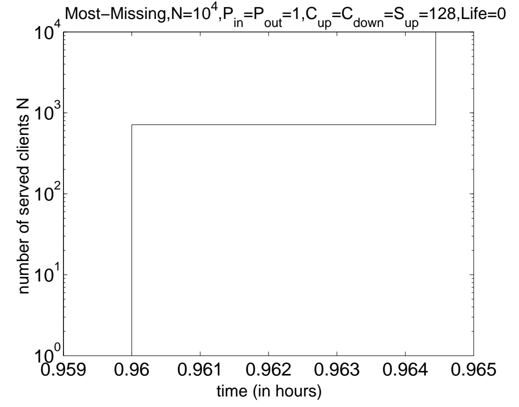


Figure 2: Number of served clients vs. time for *Most Missing*

the same time. If we compare the two architectures in absolute terms and under the parameter values given in table 1, we find that *Most Missing* needs **3472 seconds** to serve all the 10^4 clients while $PTree^k$ lasts a bit longer, **3584 seconds**[3]. 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.

This high performance of *Most Missing* is actually due to the fast distribution of chunks. In figure 3, we draw the number of copies for each chunk against time. This figure shows

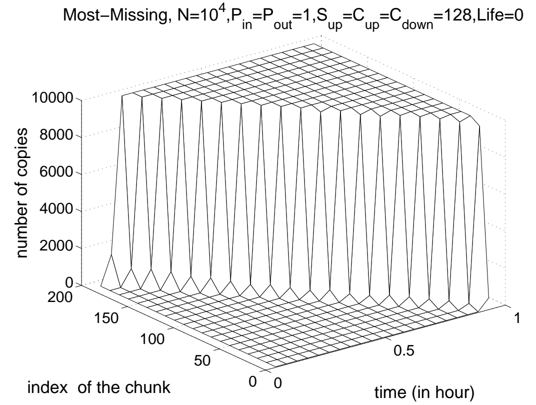


Figure 3: Chunks distribution over time for *Most Missing*

that, when a chunk is injected in the network, it gets distributed at an exponential rate. While this behavior of *Most Missing* is not intuitive, it can be easily understood through the following example. Consider the case of $N = 15$ clients and a file of $|\mathcal{C}| = 4$ chunks. 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

[3]This value is computed using eq.(6), page 7 in [1].

number of chunks, at $t = \frac{1}{|C|}$, the server schedules a new chunk C_2 to a new client. Similarly, client 1 starts delivering its chunk C_1 to another new client, say client 2. Let us focus for the moment on chunk C_1. At time $t = \frac{2}{|C|}$, client 1 uploads completely chunk C_1 to client 2. As a result, at time $t = \frac{2}{|C|}$, there are two clients, 1 and 2, that maintain a copy of the first chunk C_1. By this time $t = \frac{2}{|C|}$, 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 $\frac{1}{|C|}$ unit of time. After $\frac{i}{|C|}$ 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 $t = \frac{1}{|C|}$, i.e. $\frac{1}{|C|}$ unit of time after the first chunk C_1. By time $t = \frac{i}{|C|}$, there are 2^{i-2} clients that have chunk C_2 and only chunk C_2. More formally, at time $t = \frac{i}{|C|}$, chunk j, with $j \leq i$, has 2^{i-j} copies. Figure 4 summarizes what we are explaining. From the above analysis, we can easily verify that, at time

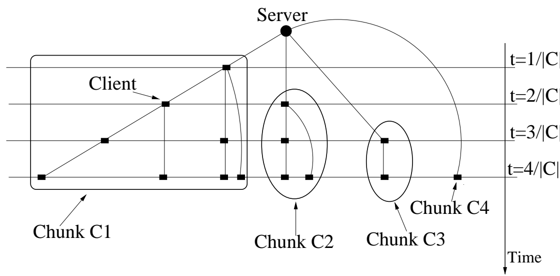


Figure 4: Growth of number of chunks for *Most Missing*

$t = \frac{4}{|C|}$, each client holds only one chunk. Clients $1, \ldots, 8$ hold chunk C_1, Clients $9, \ldots, 12$ hold chunk C_2, Clients 13 and 14 hold chunk C_3, and client 15 holds chunk C_4. By time $\frac{4}{|C|}$, two extreme scenarios can happen. The first one is that the server chooses to serve client 1. At the same time, clients $2, \ldots, 8$ exchange their chunks with clients $9, \ldots, 15$. In this case, no clients are idle and consequently, the number of copies for each chunk keeps on doubling each $\frac{1}{|C|}$ unit of time. The second scenario is that the server chooses to serve client 9 and, at the same time, clients 10, 11, and 12 exchange their chunks with clients 13, 14, and 15. Under this scenario, half of the clients, i.e., clients $1, \ldots, 8$, remain idle. In this case, the number of chunks in the system keeps increasing exponentially in time but not as fast as in the first scenario. It is hard, say impossible, to predict which scenario takes place each $\frac{1}{|C|}$ unit of time. However, we believe the behavior of *Most Missing* to be somewhere between these two extreme scenarios. A few clients remain idle while the majority exchange their chunks.

5.3 Preliminary Conclusion

In this section we studied the evolution of the number of served clients in both, mesh-based and tree-based approaches. We first compared *Least Missing* and $Tree^k$. We analytically demonstrated that, by setting the indegree of peers to $P_{in} = 1$ and their outdegree to $P_{out} = k$, *Least Missing* behaves like $Tree^k$. We then showed via simulations that *Most Missing* takes less time than $PTree^k$ to serve the same number of clients.

In addition, in our comparison, we assumed peers with homogeneous and symmetric bandwidth capacities and that there are no early departures of clients. Under this scenario, tree-based approaches exhibit their best performance. Thus, our results prove that, even under such optimal scenarios, mesh-based approaches are more efficient than tree-based ones.

6. DESIGN CHOICES FOR MESH-BASED ARCHITECTURES

In the following, we focus on mesh-based approaches and discuss the main factors that influence their performance. These factors include the peer selection strategy, the chunk selection strategy, and the network degree. For lack of space, we give results for the scenario where all clients arrive to the system at the same time. This could happen when a critical data, e.g., an anti-virus, must be updated over a set of machines as fast as possible. It can also be the case of a flash crowd where a large number of clients arrive to the system very close in time.

The simulation results that we present in this paper have been first validated under various system assumptions such as a Poisson arrival of clients and heterogeneous/asymmetric bandwidth capacity of peers. For more details, you can refer to our technical report [5].

We digress briefly to explain the simulation methodology that we use. Our 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 the link capacities (minimum of the sender upload and receiver download capacities) 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 peers according to the specified peer selection strategy. It then iterates through the sorted list until it finds a peer j that (i) needs some chunks from $D_i (D_i \cap M_j \neq \phi)$, (ii) is not already being served by peer i, and (iii) is not overloaded. We say that a peer 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 specified chunk selection strategy to choose the best chunk to send to peer j. Peer i repeats this whole process until it becomes overloaded or finds no other peer to serve.

6.1 Influence of Peer Selection Strategy

To reveal the importance of the peer selection strategy as a key design, we compare in figure 5 the scaling behavior of the two opposite strategies, *Least Missing* and *Most Missing*. The results that we plot in this figure are for the scenario depicted in table 1. Actually, this is the same scenario that we considered in section 5.2 when comparing *Most Missing* to $PTree^k$. Despite its simplicity, this scenario provides new insights while keeping the analysis extremely simple. As we can observe from figure 5, *Most Missing* performs much better than *Least Missing*. 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 5 we can also notice that *Least Missing*

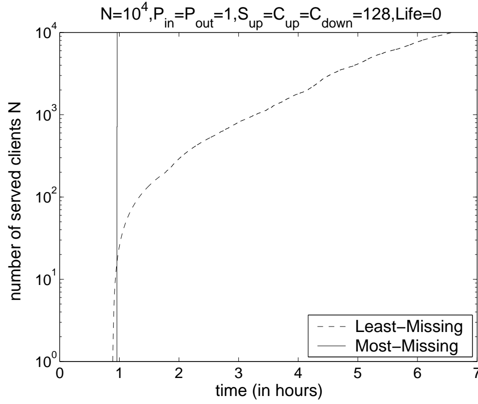


Figure 5: The number of served clients against time for *Least Missing* and *Most Missing*

optimizes the service time of the first few clients. In contrast to *Most Missing*, *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. Under the assumptions of $P_{in} = P_{out} = 1$ and $C_{up} = C_{down} = S_{up} = r$, *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 (see figure 1(b)). 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 $t = \frac{1}{|\mathcal{C}|}$, 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, i.e., *Life* = 0. 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. This means that 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.
Thus, *Least Missing* will be serving clients sequentially in a chain and the overall distribution time of the file is too slow as compared to *Most Missing*. We mention that our goal through this comparison is not to prove that *Most Missing* outperforms *Least Missing*. Instead, we tend to prove that the peer selection strategy is a key design factor that must never be ignored.

6.1.1 Conclusions

In this section we compared the two strategies *Least Missing* and *Most Missing* under a basic and homogeneous scenario. This scenario has clearly shown how the peer selection strategy 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 large delay.

6.2 Influence of Chunk Selection Strategy

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 into the system without sacrificing a lot the performance. The simulation results that we provide in this section are for the same scenario that we considered in section 5.2 (see table 1). For space reasons, we give results only for *Most Missing Random*. However, our broad conclusions also apply to *Least Missing Random*.

6.2.1 Most Missing Random

We can notice the first impact of the chunk selection strategy in figure 6 where the number of served clients with *Most Missing Random* has different scaling tendency as compared to *Most Missing*. This figure shows that, with *Most Missing Random*, around 2000 clients finish at almost the same time, within **4160 seconds** of simulation, $\sim$ 1.15 hours. Then, the number of completed clients increases slowly. Indeed, it increases by one client each $\frac{1}{|\mathcal{C}|}$ unit of time, which is equivalent to 16 seconds under the parameter values of table 1. To explain this behavior of *Most Missing Random*, we plot the chunks distribution over time in figure 7. If we take a closer look at this figure, we can observe that, after **1.15 hours** 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

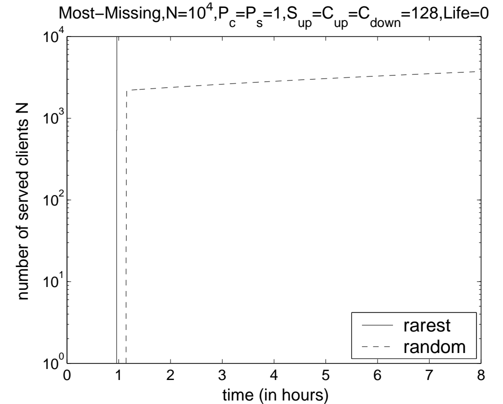


Figure 6: Impact of chunk selection strategy on *Most Missing*

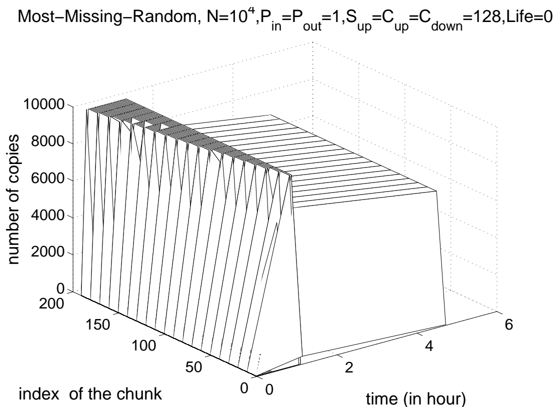


Figure 7: Chunks distribution over time for *Most Missing Random*

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 immediately. The server then delivers again this chunk C_r to a second client 2 and so on. 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**.

We give the following simplified scenario (figure 8) 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 $\frac{1}{|\mathcal{C}|}$ 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 $\frac{2}{|\mathcal{C}|}$, 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 $\frac{3}{|\mathcal{C}|}$, clients continue to exchange their chunks. For sake of simplicity, assume that clients 1 and 4 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, i.e., $Life = 0$. 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.

6.2.2 Conclusions

The chunk selection strategy is a main factor that draws

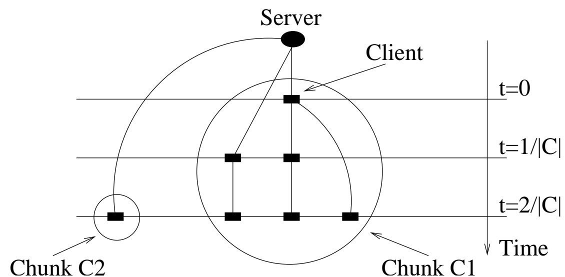


Figure 8: The distribution of the chunks over the different clients during the first $\frac{2}{|\mathcal{C}|}$ unit of time in *Most Missing Random*. We assume that the number of clients is $N = 7$ and the number of chunks is $|\mathcal{C}| = 2$

the performance of mesh-based approaches. In this section and to gain space, we evaluated only the *Most Missing* policy under the assumptions that peers schedule chunks at random and not rarest ones first. Under particular scenarios, the degradation in the system performance can be dramatic.

Fortunately, the poor performance that we saw for *Most Missing* is not only due to the chunk selection strategy. It also comes from the fact that we set $P_{in} = P_{out} = 1$ and $Life = 0$. When we increase the network degree, we lessen the likelihood that one single chunk becomes a bottleneck. Also, in practice, not all clients would be selfish and thus, the performance of the system would not degrade that much.

6.3 Influence of Network Degree

The choice of the indegree and outdegree of peers is not an easy task. A small network degree can be suitable for some architectures while it is bad for others. In this section we show through simulations how the performance of mesh-based approaches can be completely different. We increase P_{in} and P_{out} from 1 to 5 while other parameter values remain the same (see table 2).

Table 2: Parameter values

C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
128 Kbps	128 Kbps	128 Kbps	5	5	0

6.3.1 *Most Missing*

We first analyze the *Most Missing* policy with the number of completed clients graphed in figure 9. The results show again that clients download the file quickly and finish at almost the same time. What is interesting here is that, having multiple download/upload connections is not of benefit to the *Most Missing* policy.

To better understand this result, we consider in figure 10 the evolution of one single chunk in *Most Missing*. We compare the time needed to distribute this chunk to 15 clients for two different values of the network degree, i.e., $P_{in} = P_{out} = 1$ 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 $P_{in} = P_{out} = 3$. As we can observe from figure 10, when $P_{in} = P_{out} = 1$, the chunk is distributed to 15 clients within $\frac{4}{|\mathcal{C}|}$ unit of time. In contrast, when $P_{in} = P_{out} = 3$, we need $\frac{6}{|\mathcal{C}|}$ unit of time.

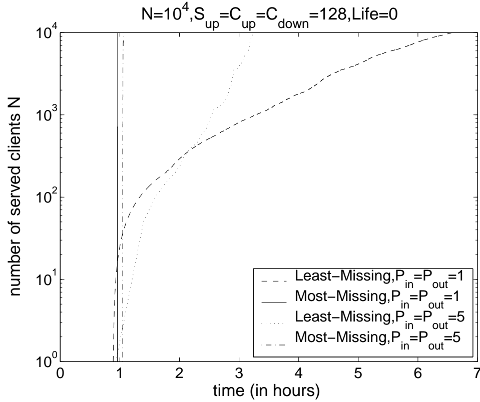
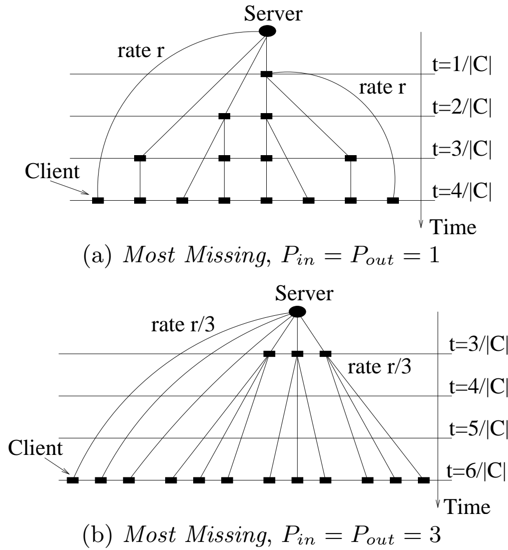


Figure 9: Impact of network degree on the system performance



(a) *Most Missing*, $P_{in} = P_{out} = 1$

(b) *Most Missing*, $P_{in} = P_{out} = 3$

Figure 10: The evolution of the number of copies of one single chunk in *Most Missing* when $S_{up} = C_{up} = C_{down} = r$

6.3.2 *Least Missing*

In contrast to what we saw for *Most Missing*, increasing the network degree improves significantly the performance of *Least Missing*. As we can observe in figure 9, for $P_{in} = P_{out} = 5$, the time needed to serve 10^4 clients is halved compared to the scenario where $P_{in} = P_{out} = 1$, i.e., **11680 seconds** instead of **23728 seconds**. This improvement in the performance of *Least Missing* is expected and in accordance with the analysis that we presented in section 5.1. There we showed that when $P_{in} = P_{out} = 1$, *Least Missing* performs poorly because it serves clients sequentially, i.e., similarly to the *Linear* architecture. In contrast, when the outdegree of peers is larger than 1, clients are served in parallel, i.e. as in $Tree^k$, and *Least Missing* becomes more efficient.

6.3.3 *Conclusions*

We investigated the impact of the network degree on our two policies. 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. This result is very interesting as it shows how the different factors interact with each other and sometimes, lead to counter-intuitive behaviors.

Regardless of what we have seen in this section, 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. Still, what is interesting to do is to derive the optimal network degree in function of the cooperation strategy that is employed.

7. DISCUSSION

In this paper, we addressed the content distribution service in P2P networks. The performance of P2P distribution architecture is drawn by two design choices. The first one deals with the way clients must organize in the network, i.e., in a tree or a mesh. In our study, we proved that mesh-based approaches are not only simpler to construct and maintain than tree-based approaches, but they also take less time to serve the same number of clients. The second design choice is related to the cooperation strategy between peers. A cooperation strategy is the result of three factors coupled together. These factors include the peer selection strategy, the chunk selection strategy, and the network degree. We investigated through simulations the influence of these factors on the performance of the system. Even though our analysis is basic, it allowed us to draw many interesting conclusions. We believe that our results and discussions present helpful guidelines for the design of future distribution architectures.

Our work can be seen as a first step towards a more complete study of P2P distribution architectures. Future work can proceed along many avenues. For instance, we need to evaluate a larger set of peer selection strategies. One could think of selecting the "most cooperating peer" or the peer that has the largest upload capacity. Also the choice of the network degree is still difficult. One interesting contribution would be to derive a relation between the network degree and the peer selection strategy.

8. ACKNOWLEDGMENTS

The authors would like to thank Dr. Arnauld Legout and Dr. Chadi Barakat for their helpful comments and discussions.

9. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] P. A. Felber and E. W. Biersack. Self-scaling networks for content distribution. In *Proc. of Self-**, Bertinoro, Italy, May 2004.

[5] A. A. Hamra. Cooperative strategies for file replication in p2p networks. Technical Report RR-04-098, Institut Eurécom, Sophia Antipolis, France, October 2004.

[6] 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.

[7] L. E. Schrage. A proof of the optimality of the shortest remaining service time discipline. *Operations Research*, 16:670–690, 1968.