UNIVERSITE DE NICE-SOPHIA ANTIPOLIS - UFR SCIENCES

Ecole Doctorale de Sciences et Technologies de l'Information et de la Communication

T H E S E

pour obtenir le titre de

Docteur en Sciences de l'UNIVERSITE de Nice-Sophia Antipolis

Discipline : Informatique

présentée et soutenue par

Anwar Al Hamra

TITRE

Approaches for Scalable Content Distribution in the Internet

Soutenue publiquement le vendredi 3 décembre 2004 devant le jury composé de

Professeur	Walid DABBOUS	président
Professeur	André-luc BEYLOT	rapporteur
Professeur	Thomas PLAGEMANN	rapporteur
Docteur	Pablo RODRIGUEZ	examinateur
Professeur	Ernst W. BIERSACK	directeur de thèse
Professeur	Guillaume URVOY-KELLER	co-directeur de thèse

Acknowledgment

First of all, I would like to express my gratitude to my thesis advisor Prof. Ernst W. Biersack for guiding me during my thesis. His continuous help and constructive remarks were invaluable for my research.

I am also grateful to my thesis co-advisor Prof. Guillaume Urvoy-Keller for his great contributions during the past three years. I am indebt to him for his everlasting encouragement.

I would like to thank David Choi for his major contribution in the development of the analytical model in chapter 2. Special thanks go to Prof. Mohamed Jarraya who contributed to the extensions given in chapter 3. I also thank Prof. Mary Vernon for her detailed comments on the work presented in chapter 4.

During my thesis, I had the opportunity to work with Prof. Pascal Felber. I gratefully acknowledge his great help in the work presented in chapter 5.

I also want to thank all people in the Corporate Department and the Institut Eurécom for their reception and support. I do not forget to acknowledge the PhD students for their friendship and specially, my colleagues Melek Onen, Fabien Pouget, and Walid Bagga, with whom I shared the office during my stay at the Institute. Special thanks to my close friends Chadi Barakat and Karim Ben Chehida for their special support during the hard period of my thesis.

Last but not the least, I owe a special thanks from all my heart to my parents, and my two brothers Wissam and Nizar, for their trust and never ended support over the several years of my studies. I am also grateful to my girlfriend Anne Duflos for her patience and enriching my life with her love.

Abstract

In this thesis we address the problem of distributing large contents in the Internet. We focus on two interesting and very common services, video on demand (VoD) and file replication.

We first investigate how to provide an efficient VoD service to a large number of clients in a dedicated overlay network (e.g. Akamai [5]). Our contribution here is a new video distribution architecture that is highly scalable and very cost effective. The novelty of our architecture is an analytical cost model that allows us to find the minimum delivery cost of the video. In contrast to previous cost models, our model includes both, the network bandwidth cost and the server cost consumed for storing and scheduling the video at the different servers.
Using the cost model, we study many interesting scenarios like the dimensioning problem of a VoD system from scratch or the evaluation of architectural choices. One architectural choice that we consider is the use of satellite to broadcast a part of the video.

We then continue our work on VoD but in a P2P network (rather than overlay network). While previous work constructs multicast trees to deliver the video to clients, we prove that we can achieve a similar efficiency in a simpler way. To this purpose, we introduce a new approach, called PBA, where clients perform locally an algorithm to find available servants from which they retrieve the video. An available servant is an existing client that has already received or is currently receiving the video and has free upload capacity.
We evaluate PBA over a wide range of scenarios and compare it to an existing multicast tree-based approach. Our results prove that PBA not only simplifies the server, but it also consumes low network resources.

Finally, 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.

Résumé

Dans cette thèse, nous étudions le problème de la distribution de contenu de grande taille sur Internet. Nous nous concentrons essentiellement sur deux services majeurs : la vidéo à la demande (VoD) et la réplication de fichiers (service de partage).

Nous étudions en premier lieu comment fournir le service VoD à un grand nombre de clients dans un réseau dédié (par example Akamai [5]). Nous proposons une nouvelle architecture très efficace à large échelle et développons un nouveau modèle analytique permettant le calcul du coût minimal pour transmettre la vidéo. Contrairement aux modèles actuels, nous considérons non seulement le coût de la bande passante du réseau, mais également le coût côté serveur pour stocker et envoyer cette vidéo.
Notre modèle analytique nous permet d'étudier plusieurs scénarios intéressants comme le problème de dimensionnement d'un système VoD ou l'évaluation des choix architecturaux, tels que la transmission partielle de la vidéo par satellite.

En second lieu, nous nous intéressons à la VoD dans un réseau pair-à-pair. Alors que la plupart des solutions actuelles utilise des arbres multicast pour transmettre la vidéo aux clients, nous montrons que nous pouvons atteindre une efficacité similaire et d'une faon plus simple en introduisant une nouvelle approche nommée PBA. Dans PBA, les clients exécutent localement un algorithme ayant pour but de trouver les pairs disponibles pour le téléchargement de la vidéo. Un pair est un client actif du réseau ayant déjà reçu ou étant en train de recevoir la vidéo.
Nous évaluons notre approche par un très grand nombre de scénarios et nous la comparons avec une approche basée sur les arbres multicast (P^2Cast). Suite à cette comparaison, nous montrons que notre approche simplifie la tâche du serveur et nécessite moins de ressources réseaux.

Enfin, nous étudions la réplication des fichiers dans les réseaux pair-à-pair. Les solutions actuelles peuvent être classées en deux catégories, selon que les clients soient organisés en arbre ou en maille. Notre première contribution consiste à comparer les deux catégories du point de vue performance à large échelle. Nous montrons que les architectures en maille sont au moins aussi efficaces que celles en arbre, tout en restant plus simples et plus dynamiques. Nous élaborons ensuite une analyse complète des approches en maille en identifiant les principaux paramètres qui influent sur la performance du système. Cette analyse permet, pour des scénarios donnés, de concevoir aisément de nouvelles architectures dédiées.

Contents

List of Parameters

Parameters for chapters 2 and 3	
m	Tree breadth
l	Tree depth
τ_j	Active occupation duration for link at depth j (prefix transmission)
$P(j)$	Prob. that link at depth j is active (suffix transmission)
h	Prefix server height
L	Video length (min)
D	Prefix length (min)
$L - D$	Suffix length (min)
N_s	Number of segments of the suffix $(N_s \triangleq \lceil \frac{L-D}{D} \rceil)$
b	Consumption rate of the video [Mbps]
K	Number of videos in the system
λ	Average number of requests per min
N	Average number of clients
T	Threshold time (min)
C	Scalar constant

Parameters for chapter 4	
S_p	Server placement
T_s	the maximum amount of time a client can stay in the network
T_{cast}	Threshold for P^2Cast
T_{pba}	Threshold for PBA
L_{bw}	Bandwidth capacity of local area networks
BW_{tt}	Link between two transit-domain nodes
BW_{ts}	Link between one transit-domain node and one stub-domain node
BW_{ss}	Link between two stub-domain nodes
C_s	Number of servants a client can contact

N_a	Average number of admited clients

Parameters for chapter 5

$\mathcal{C}$	Set of all chunks the file is divided into
$\|\mathcal{C}\|$	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

Chapter 1

Introduction

1.1 Content Distribution in the Internet

The problem of content distribution in the Internet has been subject for an intensive study in the last decade. Nowadays, there exist two popular and widely used solutions for this problem:

- Dedicated overlay networks that deploy machines in the core of the Internet to the purpose of storing and delivering the content. One example is Akamai [31], which advertises 1000 customer sites and handles around 15% of the Internet traffic [1].

- Peer-to-Peer (P2P) networks, a new solution that relies on clients to distribute the content. By capitalizing the upload capacity available at the clients, P2P networks offer great potential for addressing two of the most challenging issues of today's Internet: The cost-effective distribution of bandwidth-intensive content to thousands of simultaneous clients and the resilience to flash crowds[1].

Given these solutions, one main issue is how to schedule the content in order to efficiently leverage the available resources and satisfy the clients demands. **Approaches for Scalable Content Distribution in the Internet** is the topic of this thesis. As one knows, there is no approach that can accommodate all services. Each service has its requirements and its goals. In this thesis, we focus on two services with high system and network resources requirements. These two services are the video on demand and the file replication:

- With the increasing availability of high access bandwidth of clients, video on demand (VoD) applications are now gaining popularity. Applications for VoD include, e.g., online training, news broadcasts, educational programming. However, the intensive-bandwidth and long live nature of video content make a large scale VoD service over the Internet a challenging problem. In this thesis, we address the VoD service in both, dedicated overlay and P2P networks. For each scenario, we introduce a new video distribution architecture that ensures efficiency and scalability.

- File replication, on the other hand, represents another important service that allows clients to retrieve the file they want from the Internet. As files can refer to any kind of content, we are mainly interested in non real-time content. Moreover, we focus on the file replication in P2P networks. While file sharing has been for a long time the only P2P application, P2P file replication is now becoming a dominant service. According to the *AlwaysOn* site [7], *BitTorrent*[2] alone accounts for

[1]We mean by a flash crowd a huge and sudden surge of request traffic that usually leads to the collapse of the affected server, as happened to the Web sites of major news companies during the event of September 11th.

[2]*BitTorrent* is a popular P2P tool for file replication in the Internet that we describe later on.

53% of the overall P2P traffic. Our contribution here is a deep analysis that represents guidelines for the design of new architectures for this service.

1.1.1 Organization of this Chapter

The rest of this chapter is structured as follows. In sections 1.2 and 1.3, we review related work on VoD and file replication. Section 1.3 summarizes the contributions of this thesis and section 1.5 describes briefly the content of the following chapters.

1.2 Related Work on VoD

1.2.1 VoD in Dedicated Overlay Networks

VoD in dedicated overlay networks has been extensively investigated over the years. Existing schemes lie on multicast to achieve scalability and can be largely classified into *open-loop* and *closed-loop* schemes.

- Open-loop schemes partition each video into smaller pieces called *segments* and transmit each segment at its assigned transmission rate. The first segment is transmitted more frequently than later ones because it is needed first in the playback. In open–loop schemes there is no feedback from clients to the server and the transmission is completely one–way: all segments are broadcast regardless of the number of clients in the system. The most important property of open-loop schemes is that the server bandwidth cost is independent of the number of clients requesting the video. However, their main drawback is that they introduce a start-up delay[3] and waste bandwidth in the case of non-popular videos.

- Closed-loop schemes, on the other hand, require the client to contact the server. The server initiates a new stream each time a new client or a set of clients request the video. In most cases, a new client joins an ongoing multicast stream that has been initiated for earlier clients, if there is any, and retrieves the missed part due to its late arrival via unicast. The majority of closed-loop schemes ensure an immediate playout of the video. Usually, these schemes are suitable for videos with low and moderate popularities where the load on the server is reasonable.

Open-loop Schemes

The basic open-loop scheme is *Staggered Broadcasting* proposed by Almeroth et al. [12]. In this scheme, the server allocates for each video C channels each of bandwidth b, where b is the playback rate of the video expressed in Mbps (see table 1.1 for notations used in this thesis). On each channel, the whole

Notation	**Definition**
MB	Mega bytes
Mb	Mega bits
Mbps	Mega bits per second
min	minute

Table 1.1: Notations used in this thesis.

video is broadcast periodically at its full rate b. The starting transmission points on the different channels are staggered to guarantee a start-up delay of at most $\frac{L}{C}$ min. The parameter L represents the length of

[3]The start-up delay denotes the maximum time a client may wait before playing out the video. Note that we ignore here the *transmission* delay due to sending a request to a server or joining a multicast group.

the video expressed in min. Clients in *Staggered Broadcasting* download from only one channel at the same time and no local storage capacity is needed. The start-up delay can be improved only with a linear increase in the number of channels C.

Viswanathan et al. [90] propose an original idea to reduce the start-up delay while avoiding the linear increase in the number of channels as in *Staggered Broadcasting*. This is done as follows. Given the server bandwidth capacity B (in Mbps) and the number of videos K to be broadcast, the authors advise to multiplex the K videos together on C channels each of bandwidth $\frac{B}{C}$. Then, each video is split into C segments and the size of each segment is made δ times larger than the previous one ($L_i = \delta \cdot L_{i-1},\ 1 < i \leq C$). Segments i for all K videos are multiplexed into the same channel i and broadcast consecutively and periodically at rate $\frac{B}{C}$. This means that, to access segment i, a client may wait up to $(\frac{K \cdot L_i \cdot b \cdot 60}{\frac{B}{C}})/60 = \frac{K \cdot b \cdot C \cdot L_i}{B}$ min, where $L_i \cdot b \cdot 60$ is the amount of data (in Mb) in a segment of length L_i min and $\frac{B}{C}$ is the broadcast rate of the different segments on channel i. Segments of the required video are downloaded in order and only from the beginning. The client starts downloading segment 1 at the first occurrence and starts consuming it concurrently. For the subsequent segments, once the client begins consuming segment $(i-1)$, it listens to the ith channel to download segment i. To avoid interruption while playing out the video, the client should start downloading segment i before it entirely consumes segment $(i-1)$. Mathematically speaking, $L_{i-1} \geq \frac{K \cdot b \cdot C \cdot L_i}{B}\ \forall\, 1 < i \leq C$, where L_{i-1} min is the time needed to consume segment $(i-1)$ and $\frac{K \cdot b \cdot C \cdot L_i}{B}$ min is the maximum waiting time before start receiving segment i. By substituting L_i by $\delta \cdot L_{i-1}$, we obtain $1 < \delta \leq \frac{B}{K \cdot b \cdot C}$.
This scheme, referred to as *Pyramid Broadcasting* (PB), reduces significantly the server bandwidth and the start-up delay as compared to *Staggered Broadcasting*. The authors suggest to set δ to its maximum value (i.e. $\delta = \frac{B}{K \cdot b \cdot C}$), which makes the first segment as short as possible and consequently, minimizes the start-up delay ($\frac{K \cdot b \cdot C \cdot L_1}{B}$ min). The main drawback of PB is that segments are transmitted on the different channels at a high rate (i.e. $\frac{B}{C}$), which requires a high download capacity at the client side.

Aggarwal et al. in *Permutation-based Pyramid Broadcasting* (PPB) [10] relax the high download requirement at the clients in PB at the expense of a larger start-up delay and a more complex synchronization. PPB uses the same geometric series proposed by PB to define the length of each segment ($L_i = \delta \cdot L_{i-1},\ 1 < i \leq C$). The main difference between the two schemes is that PPB proposes to further divide each channel into $K \cdot p$ sub-channels, where p sub-channels are dedicated to the same video segment. Then, each segment is broadcast on p sub-channels at rate $\frac{B}{C \cdot K \cdot p}$. For each video, the transmission points of segment i on the different p sub-channels are shifted in such a way that the access time to that segment is at most $\frac{K \cdot b \cdot C \cdot L_i}{B}$ min. In contrast to PB, a PPB client starts downloading segment i once segment $(i-1)$ is entirely received. The condition $\delta \leq (\frac{B}{K \cdot b \cdot C} - p)$ with $\delta > 1$ ensures no interruptions in the display of the video.
As compared to PB (i) PPB reduces greatly the download capacity required at the client side. A PPB client downloads from a single channel at a time at rate $\frac{B}{C \cdot K \cdot p}$ instead of $\frac{B}{C}$ and (ii) PPB assigns to δ a higher value that makes the first segment larger, which in turn increases the start-up delay.

To achieve both, a low start-up delay (as in PB) and a reasonable download capacity at the clients (as in PPB), *Skyscraper Broadcasting* (SB) [49] proposes to transmit each of the K videos separately. Given the server bandwidth B and the number of videos K, the server allocates a bandwidth of $\frac{B}{K}$ to each video. Then, the bandwidth dedicated to each video is split into $C = \lfloor \frac{B}{K \cdot b} \rfloor$ channels each of bandwidth b.

SB uses a recursive function instead of a geometric series to generate the segment lengths of each video:

$$L_i = \begin{cases} L_1 & \text{if } i = 1 \\ 2 \cdot L_1 & \text{if } i = 2, 3 \\ 2 \cdot L_{i-1} + 1 & \text{if } i \text{ modulo } 4 = 0 \\ L_{i-1} & \text{if } i \text{ modulo } 4 = 1 \\ 2 \cdot L_{i-1} + 2 & \text{if } i \text{ modulo } 4 = 2 \\ L_{i-1} & \text{if } i \text{ modulo } 4 = 3 \end{cases} \tag{1.1}$$

In addition, SB introduces a new parameter W to restrict segments from becoming too large. From equation 1.1, when the size L_i of segment i exceeds $W \cdot L_1$, L_i is set to $L_i = W \cdot L_1$. The following series $[1, 2, 2, 5, 5, 12, 12, 25, 25, 25, 25 \dots]$ represents an example for $W = 25$ and $L_1 = 1$ min. A SB client listens to at most two adjacent channels simultaneously and downloads the corresponding segments at the first occurrence, which guarantees a start-up delay of no more than L_1 min.

Just after *Skyscraper Broadcasting*, Juhn et al. [53] initiate *Harmonic Broadcasting* (HB), a new family of open-loop schemes. HB divides the video into n segments of equal size. Then, each segment i is equally broken up into i sub-segments $(i, 1), \dots, (i, i)$, which are broadcast consecutively and repeatedly on channel i at rate $\frac{b}{i}$. Under these assumptions, segment 1 will consist of one single sub-segment $(1, 1)$ and would be broadcast at rate b.
When a new client arrives, it first listens to channel 1. When a new transmission of segment 1 begins, the client starts downloading from all channels concurrently and, at the same time, starts playing the video. Given the start-up delay to be achieved, HB requires the lowest server bandwidth as compared to the above schemes. For a start-up delay of $\frac{L}{n}$, the server bandwidth is around $b \cdot \log(n)$. Since clients download from all channels simultaneously, this scheme demands from clients a download capacity of $b \cdot \log(n)$.
However, Pâris et al. [70] have shown that HB does not guarantee to deliver all segments in time. As an example, consider a client that arrives at time t_0 as depicted in figure 1.1. Figure 1.1 illustrates the trans-

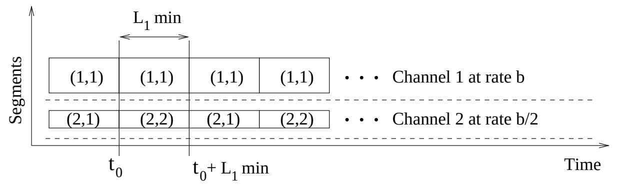


Figure 1.1: The transmission process on the first two channels in HB.

mission process on the first two channels in HB. At time t_0, the client starts receiving and consuming the first segment. Meanwhile, it stores sub-segment $(2, 2)$ at rate $\frac{b}{2}$. L_1 min later, the client consumes entirely segment 1 and must start playing out sub-segment $(2, 1)$. Yet, by this time, the client will be playing sub-segment $(2, 1)$ at rate b while receiving it at rate $\frac{b}{2}$, which causes an interruption. To prevent this interruption, HB requires an extra start-up delay; when a client starts receiving the first segment, it must wait $\frac{(n-1) \cdot L_1}{n}$ additional min before it starts playing out the video. To avoid this extra waiting time, Pâris et al. propose two extensions of HB, namely *Cautious Harmonic Broadcasting* (CHB) and *Quasi-Harmonic Broadcasting* (QHB), where they require the server to use more bandwidth.

In addition to CHB and QHB, the authors develop a new variant of HB, called *Polyharmonic Broadcasting* (PHB) [71]. The novelty of PHB is a fixed start-up delay for all clients regardless of their arrival time. In other words, as compared to HB, the main difference in PHB is that the client starts playing out the video once it has been in the network for L_1 min, even if the first segment has been available before.

Later on in [52], the same authors of HB (Juhn et al.) introduce with a new and efficient broadcasting scheme. This scheme, referred to as *Fast Broadcasting* (FB), divides equally the server bandwidth amongst the K videos to be broadcast. For each video, it dedicates $\lfloor \frac{B}{K \cdot b} \rfloor$ channels each of bandwidth b. Given the values of B and K, FB ensures the lowest start-up delay as compared to the previous schemes. FB partitions each video into $(2^{\lfloor \frac{B}{K \cdot b} \rfloor} - 1)$ segments of equal size. Channel i transmits periodically and repeatedly segments $2^{i-1}, \ldots, (2^i - 1)$ and clients download from all channels at the same time. The advantage of FB is that it makes the first segment very small, which in turn reduces the start-up delay[4]. However, this scheme requires clients to have a high download capacity up to $b \cdot \lfloor \frac{B}{K \cdot b} \rfloor$.

In addition to the mentioned schemes, there are hybrid ones that combine many techniques together. For instance, the *Pagoda Broadcasting* PGB and the *New Pagoda* proposed by Pâris et al. [68, 69] combine HB and PB. The motivation behind this combination is to achieve a low server bandwidth (i.e. as in HB) while keeping small the number of channels (i.e. as in PB). The key idea is to partition each video into a large number of segments of equal size and map them into a small number of channels of equal bandwidth. They use time-division multiplexing to ensure that each segment is received in time.

These schemes that we have described so far are constrained to highly regular designs which limit their flexibility. *Tailored Periodic Broadcast* [19] is a more flexible scheme that can be adapted to meet different constraints in the system such as limited I/O capacities of the server/clients or limited storage capacity at the client side. This scheme will be detailed more in section 2.2.4. For details on open-loop schemes, one would refer to [47, 54].

Since clients can not watch the video immediately, open-loop schemes can only provide a near VoD service[5]. In contrast, most closed-loop schemes ensure a true VoD service (i.e. zero start-up delay), but at the expense of a higher load on the server.

Closed-loop Schemes

Closed-loop schemes serve the video in response to client requests. A simple example is the *Batching* scheme proposed by Anderson [13] and extensively studied in [28, 39, 9]. The main idea here is to batch clients that arrive within a given interval of time, in order to serve them via a single multicast stream. In *Batching*, clients download from one single stream at rate b and no local storage capacity is needed. The main limitation of this scheme is that its performance is proportional to the start-up delay of the clients.

Another example is *Patching* that, as opposed to *Batching*, appears to be efficient and at the same time ensures a zero start-up delay. *Patching* has been proposed by Hua et al. [48] and evaluated afterwards by Cai et al. [20]. In *Patching*, the first client that requests a video receives a complete stream for the whole video from the server. When a new client arrives after the first one, we distinguish two cases:

- The complete stream that has been initiated for the first client is still active. In this case, the new client needs to connect to that stream which changes from unicast to multicast. In addition, the new client receives immediately from the server a unicast patch for the part it missed in the complete stream due to its late arrival.
- There is no active complete stream. In this case, the server initiates a new stream and the above process is repeated for clients that arrive later.

As compared to *Batching* techniques, *Patching* requires clients to have a larger download capacity. Clients may join two streams simultaneously each at rate b.

[4]The start-up delay in FB is equal to the length of the fi rst segment, $\frac{L}{(2^{\lfloor \frac{B}{K \cdot b} \rfloor} - 1)}$ min.

[5]An idea to avoid this start-up delay is to preload in advance some of the fi rst segments of the video into a set-top box at the clients [72]

Patching has been afterwards extended by Gao et al. [37] to produce *Controlled Multicast*. The novelty of *Controlled Multicast* is a threshold policy to reduce the cost of the unicast patches. Whenever a new client arrives and a complete stream is active, the threshold value serves to decide whether to initiate a new complete stream for that client, or whether that client must join the last ongoing one. This scheme achieves a server cost that increases with the square root of the number of requests O($\sqrt{\lambda \cdot L}$), given a Poisson arrival of clients with rate λ.

Eager et al. [34] derive the minimum server bandwidth needed to serve N clients and, at the same time, achieve an instantaneous playout of the video; the server cost is $b \cdot \log(N+1)$ when the arrival process is Poisson. In addition, the authors describe the *Hierarchical Multicast Stream Merging* scheme (HMSM) [35] that achieves a near optimal server bandwidth cost. As its name indicates, this scheme merges clients in a hierarchical manner. When a new client arrives, the server initiates an unicast stream to that client. At the same time, the client listens to a target stream that is still active[6]. When the client receives via unicast all what it misses in the target stream, the unicast stream is terminated and the client merges into the target stream, and the process repeats.

We note that the idea of merging itself is not new. Golubchik et al. [41] have already addressed this point and proposed a scheme called *Adaptive Piggyback*. This scheme alters the display rate of requests in progress for the purpose of merging their respective video streams into one single stream. Consider the following example where client 1 is receiving a video stream from the server at rate b. Some time later, a new client 2 requests the same video and a new stream is initiated from the server. To merge the two streams, the server slows down the playback rate of the first stream and speeds up the rate of the second one until both streams merge into one single stream. However, the performance of this kind of merging is mainly limited by the fact that the variation in the playback rate of the video must be within, say $\pm 5\%$ of the normal playback rate, or it will result in a perceivable deterioration of the playback quality.

There exist also hybrid schemes that combine *Batching* and *Patching* techniques. One example is the OBP scheme (*Optimized Batch Patching*) proposed by White et al. [92]. As for the classical *Patching* [48], in OBP, A new client (i) Either receives a new complete stream in case there is no active one or (ii) Joins the most recent one. In contrast to *Patching*, in case the new client joins an ongoing complete stream, it does not receive immediately a unicast patch for what it missed in that stream. Instead, the server batches many clients that arrive within a given interval of time and serves them via multicast. In addition, the authors extend OBP to deal with the case of heterogeneous clients where each client chooses the start-up latency according to the price it is willing to pay for the service.

Another hybrid scheme is *Mcache* proposed by Ramesh et al. [74]. *Mcache* splits the video into two parts, the prefix and the body. The prefix is stored locally at the client side in a set-top box while the body is stored at the server side. This combination ensures a zero start-up delay and allows the server to batch requests for the body of the video for the duration of the prefix. The requests batched together are then served via one single multicast stream. When a client arrives after the multicast stream has begun, it joins the rest of that stream and receives what it missed via a patch. As in *Controlled Multicast*, the authors associate a threshold to figure out when to start a new multicast stream.

From the above discussion, we can conclude that:

- The main advantage of open-loop schemes is that the server bandwidth cost is constant and independent of the number of clients in the system. This property makes these schemes very efficient for the transmission of popular videos. Yet, open-loop schemes waste bandwidth in case of non popular videos and introduce a start-up delay.

- In contrast, most closed-loop schemes ensure a zero start-up delay. In these schemes, each new

[6]The choice of the target stream is not an easy task. One of the candidate solutions is to choose the closest (i.e. most recent) active stream

client contacts the server to retrieve the video, which makes the server a potential bottleneck. As a result, closed-loop schemes are only suitable for videos with low and moderate popularity.

So, one interesting idea would be to combine both schemes together. This combination ensures a zero start-up delay and makes the system suitable for both popular and non popular videos. Guo et al. [44] have developed the principles to achieve such a combination. The main idea is to divide the video into two parts, namely the prefix and the suffix, with the prefix being completely viewed before the suffix. The prefix is delivered to the clients via a closed-loop scheme while the suffix is broadcast using an open-loop scheme. This is referred to as *Prefix Caching Assisted Periodic Broadcast* (see section 2.2.1).

Our Contribution

In this thesis (chapters 2 and 3), we investigate how to provide an efficient and scalable VoD service to a large client population. We introduce a new video distribution architecture that is highly scalable, very cost effective and, at the same time, ensures a zero start-up delay. As the *Prefix Caching Assisted Periodic Broadcast* framework does, our architecture combines both open-loop and closed-loop schemes. Given this architecture, we develop a complete cost model that allows us to minimize the operational cost of the service.
Our analytical model, along with some extensions, allows us to explore several scenarios: (i) Long videos of 90 min (movies), (ii) Short videos of a few min (clips), (iii) The dimensioning of a video on demand service from scratch, (iv) The case of the optimization of an already installed video on demand service (i.e. the limited resources scenario), (v) The use of satellite to broadcast a part of the video, and (vi) Allowing clients to store locally a part of some/all popular videos.

1.2.2 VoD in P2P Networks

Most of the existing work on video streaming in P2P networks focuses on live streaming. The main idea is to distribute the video through multicast trees rooted at the server. Constructing a multicast tree for video distribution is very challenging as clients that are part of the tree may leave at any time, which may disrupt the video reception of the clients that are downstream while the tree is re-built. To prevent disruption, *CoopNet* [66] splits the video into k sub-streams using multiple description encoding techniques [42]. It then builds a multicast tree to deliver each sub-stream. In this case, the departure of a client does not interrupt the reception of all sub-streams.

Castro et al. [21] extend *CoopNet* to *SplitStream*. The originality of *SplitStream* is that each of the k sub-streams is delivered on a **distinct multicast tree** rooted at the server; a client being an interior node in one tree and a leaf node in the remaining ones. As a result, when a client leaves the network, one single sub-stream is disrupted.

There has been also a series of papers that optimize the organization of the clients in the multicast trees with respect to some metrics [25, 51, 14]. One example is the *Narada* protocol proposed by Chu et al. [25]. *Narada* adapts dynamically the multicast tree in such a way that the delay between any two clients does not exceed C times the unicast delay between them, where C is a scalar constant that depends on the outdegree of the clients. The outdegree of a client stands for the number of simultaneous upload connections the client can maintain.
Another example is *Overcast* proposed by Jannotti et al. [51]. In this protocol, clients are self-organized so as to maximize the throughput achieved between them and the server[7].
There is also the *Nice* protocol introduced by Banerjee et al. [51]. The central idea of *Nice* is to assign clients that are close to each others to the same cluster[8]. Each cluster has a leader that serves it. The

[7]in this thesis, the term throughput refers to the amount of bytes received during a specifi ed interval of time.
[8]The distance metric between clients can be chosen according to the desirable goal.

leaders of all clusters are then grouped into further clusters and so on. As a result, clients will be organized in a hierarchical manner as in a tree.

Still, few approaches have dealt with on-demand video distribution in P2P networks. *Chaining* [85] is one approach which organizes clients into chains. Clients that arrive within a given interval of time T are served with the same chain. Within the same chain, the first client 1 is served by the server. Client 2 is served by client 1 and so on. The first client to arrive after the interval of time T has expired receives a new video stream from the server. Then, a new chain is initiated and the same process is repeated.

Guo et al. [45] describe a scheme called P^2Cast that applies *Controlled Multicast* [37] to P2P systems. P^2Cast patches clients that arrive within the same interval of time into the same multicast stream, referred to as the base stream, that is transmitted over a multicast tree built on top of the clients. When a new client arrives, it joins the multicast tree for the rest of the base stream and receives a unicast patch for the part it misses in that stream due to its late arrival. As in *Controlled Multicast*, the authors associate a threshold T_{Cast} to decide when to start a new base stream. P^2Cast will be detailed more in section 4.3.1.

Our Contribution

Usually multicast trees require the server to run complex algorithms to construct and maintain the trees. While a multicast tree seems to be the optimal and intuitive solution for live streaming, we argue that this is not the case for on-demand video where clients are **asynchronous**. Our contribution here is a new pull-based approach, denoted as PBA, for an efficient VoD service in P2P networks (chapter 4). Through PBA, our goal is to show that we can achieve efficiency and scalability while keeping the server simple, i.e. without constructing multicast trees.

1.3 File Replication in P2P Networks

File replication in P2P networks is becoming very popular and there already exist approaches that provide such a service. One example is *BitTorrent* [26] developed by Bram Cohen, which is currently adopted by many web sites. The sole objective of *BitTorrent* is to quickly *replicate* a single large file to a set of clients. The challenge is thus to maximize the speed of replication.

A torrent consists of a central component, called *tracker* and all the currently active clients. *BitTorrent* distinguishes between two kinds of clients depending on their download status: clients that have already a complete copy of the file and continue to serve other clients are called *seeds*; clients that are still downloading the file are called *leechers*. The tracker is the only centralized component of the system. The tracker is not involved in the actual distribution of the file; instead, it keeps meta-information about the clients that are currently active and acts as a rendez-vous point for all the clients of the torrent.

A new client joins an existing torrent by downloading a *torrent* file (usually from a Web server), which contains the IP address of the tracker. To initiate a new torrent, one thus needs at least a Web server that allows to discover the tracker and an origin server or initial seed with a complete copy of the file. To update the tracker's global view of the system, active clients periodically (every 30 minutes) report their state to the tracker or when joining or leaving the torrent. Upon joining the torrent, a new client receives from the tracker a list of neighbors (active clients) to connect with. Typically, the tracker provides 50 neighbors chosen at random among active clients while the new client seeks to maintain connections to $20-40$ neighbors. However, only a part of these connections are active at a time, i.e. is used for uploading/downloading the data. If ever a client fails to maintain at least 20 connections, it recontacts the tracker to obtain additional neighbors. The set of neighbors to which a client is connected is called its *neighbors set*.

The clients involved in a torrent cooperate to replicate the file among each other using *swarming* techniques: the file is broken into equal size chunks (typically 256 kB each) and the clients in a neighbors set exchange chunks with one another. The swarming technique allows the implementation of parallel download [78] where different chunks are simultaneously downloaded from different clients. Each time a client obtains a new chunk, it informs all the neighbors it is connected with. Interactions between clients are primarily guided by two principles. First, a client preferentially sends data to neighbors that reciprocally sent data to it. This "tit-for-tat" strategy is used to encourage cooperation and ban "free-riding" [8]. Second, a client limits the number of neighbors it serves simultaneously to 4 neighbors and continuously looks for the 4 best downloaders (in terms of the rate achieved) if it is a seed or the 4 best uploaders if it is a leecher.

BitTorrent implements these two principles, using a "choke/unchoke" policy. "Choking" is a temporary refusal to upload to a neighbor. However, the connection is not closed and the other party might still upload data. A leecher services the 4 best uploaders and chokes the other neighbors. Every 10 seconds, the leecher re-evaluates the upload rates for all the neighbors that transfer data to it. There might be more than 4 neighbors uploading to it since first, choking is not necessarily reciprocal and second, clients are not synchronized[9]. The leecher then chokes the neighbor, among the current top 4, with the smallest upload rate if another neighbor offered a better upload rate. Also, every 3 rounds, that is every 30 seconds, a client performs an *optimistic unchoke*, and unchokes a neighbor regardless of the upload rate offered. This allows to discover neighbors that might offer a better service (upload rate). Seeds essentially apply the same strategy, but based solely on download rates. Thus, seeds always serve the clients to which the download rate is highest.

Another important feature of *BitTorrent* is the chunk selection algorithm. The main objective is to consistently maximize the entropy of each chunk in the torrent. The heuristic used to achieve this goal is that a client always seeks to upload the chunk the least duplicated in its neighbors set (keep in mind that clients only have a local view of the torrent). This policy is called the *rarest first policy*. There exists an exception to the rarest first policy when a new client joins a torrent and has no chunks. Since this new client needs to quickly obtain a first chunk (through optimistic unckoke), it should not ask for the rarest chunk because few clients hold this chunk. Instead, a new comer uses a random first policy for the first chunk and then turns to the rarest first policy for the next ones.

We have extensively analyzed a large torrent (with up to a few thousands of simultaneous active clients) over a period of five months [50]. The performance achieved, in terms of the throughput per client during download, and the ability to sustain a high flash-crowd, demonstrate that *BitTorrent* is highly effective.

A solution similar to *BitTorrent* is *Slurpie* introduced by Sherwood et al. [84]. *Slurpie* aims at enforcing cooperation among clients to alleviate the load at the server that is the primary source of the content. The algorithms of *Slurpie* are much more complex than the ones of *BitTorrent* and require to estimate the number of active clients in the *Slurpie* network. The expected improvements over *BitTorrent* is less load on the *topology server* (equivalent to the *BitTorrent* tracker) and on the server through a back-off algorithm.

As *BitTorrent* does, *Slurpie* divides each file into many chunks of 256 KB each. When a new client wants to download a file, it first registers at the well known topology server and gets a list of the ϕ most recent clients that have requested the file. Thus, in contrast to the tracker in *BitTorrent*, the topology server need not keep track of all active clients in the network. In addition, clients report no information about their status to the topology server. These ϕ chosen clients represent the initial set of neighbors of the new client, which will be updated as new neighbors are discovered or existing ones leave the network. Upon receiving its list of neighbors, the new client connects to a random sub-set of these ϕ neighbors.

[9]For instance, a client that offered a very good upload rates has decided to choke this connection just 1 second before the reevaluation on the other side and thus it might still be in the top 4 list for the next 10 seconds and received service.

A client typically seeks to maintain $\eta \geq O(\log N)$ connections where N is the number of clients in the network. Along all connections, update information are exchanged between clients to know which client has what chunk. The rate at which a client receives such update information increases additively and decreases multiplicatively depending on the estimated available bandwidth at the client[10]. This decision is motivated by the fact that, when a client does not know where to find the chunks it misses, it needs more update information to discover new neighbors and therefore increases the rate at which it receives these information. Correspondingly, when the download of the client is fully utilized, the client needs less information about other clients. Amongst the set of connections a client maintains, few of them are used to download the file; the number of simultaneous download connections also increases additively and decreases multiplicatively with the available bandwidth at the client.

The novelty of *Slurpie* is a back-off algorithm to reduce the load on the server. When a client misses a chunk that none of its neighbors holds, it decides to download that chunk from the server with a probability of $\frac{C}{N}$, where C is a scalar constant. Thus, at any moment, there will be on average C simultaneous connections to the server. Results are promising since *Slurpie* is able to outperform *BitTorrent* in a controlled environment. Still, the actual performance of *Slurpie* in case of flash crowds and for a large number of clients is unknown.

FastReplica (FR), developed by Cherkasova et al. [23], addresses how to efficiently replicate a large file over a set of N clients that are assumed to be known and stable. The file is divided into N chunks of equal size. In a first step, the server sets-up N connections to all N clients in the system. Over each connection, the server transmits a disjoint chunk, i.e. chunk C_i to client i. When client i receives chunk C_i, it opens $N-1$ simultaneous connections with all remaining clients in the system to deliver that chunk.

While FR was basically designed for small number of clients, $N = 10-30$, the authors describe how it can be extended to scale when N is large. That is, the set of clients is divided into groups of size C each, where C is in the order of $10-30$. In this case, the number of chunks is also equal to C. Then, FR is executed iteratively. During the first iteration, the file is replicated over a first group, i.e. C clients. Once a client receives completely the file, it opens C new connections to C new clients and the same process is repeated. As a result, after the second iteration, C^2 new clients are served, C^3 new clients after the third iteration and so on until all clients get served.

The main limitation of FR is that it assumes homogeneous bandwidth capacities of clients and homogeneous connections with the same transfer rate. Under these assumptions and in a real environment, the overall replication time depends on the replication time of the chunk traversing the worst connection in terms of transfer rate. Lee et al. [58] have extended FR to produce *Adaptive FastReplica* (AFR) for heterogeneous environments. The key idea of AFR is to divide the file into chunks with different sizes. Large chunks are transmitted over connections that provide high transfer rates and vice versa. To learn about the available bandwidth over each connection, AFR requires clients to report to the server the transfer rates that they achieve over the different connections they maintain. At the beginning, the server has no prior knowledge of the network status and therefore, it starts with an arbitrary division of the file. As the algorithm is executed iteratively (i.e. in case of a large number of clients) or as more files are replicated, the server refines its information about the connection capabilities and adapts the sizes of the different chunks in order to maximize the throughput of the system.

Another related approach is *PROOFS* proposed by Stavrou et al. [87]. *PROOFS* is a completely randomized approach that was basically designed to handle flash crowds. When a new client arrives to the network, it contacts a well known server and gets an initial set of neighbors, i.e. set of existing clients. Clients update continually their set of neighbors by performing a shuffle operation as follows. A shuffle is executed between two clients and can be initiated by any client. Consider client 1 that wants to start

[10]Each client estimates its available bandwidth once every second. This is done by simply measuring the number of bytes received over all connections.

a shuffle. Client 1 selects at random a sub-set of neighbors out of its complete set. From this sub-set, it then selects a random neighbor, say client 2, and asks it to participate into the shuffle. If client 2 rejects the shuffle request, the client waits a random amount of time before it starts a new shuffle. In contrast, if client 2 accepts to participate into the shuffle, it receives from client 1 the sub-set of neighbors this client has selected. Similarly, client 2 selects a sub-set of neighbors and sends it to client 1. Upon receiving each other's sub-set, the two clients add the new neighbors to their set as follows: (i) No neighbor appears twice within the same set, (ii) A client is never its own neighbor, (iii) New neighbors replace old ones only when the size of the set exceeds a constant C. Note that if client 2 does not answer to the shuffle request of client 1, client 1 considers that client 2 has left the network and deletes it from its set of neighbors.

When a client wants to retrieve a file, it initiates a query with: (i) The requested file, (ii) A TTL that counts for the maximum number of times the query gets forwarded, and (iii) A fanout f to indicate to how many neighbors a client can forward the query. Upon receiving a query for a file, the client first checks whether it holds the file. If this is the case, the client unicasts directly the file to the client that initiated the request. Otherwise, the client decrements the value of the TTL and forwards the query to f of its neighbors.
The performance evaluation of the approach has mainly focused on the time needed and the probability to find a file. The results exhibit a low latency and a very high probability.

The approaches that we described so far all organize clients in a mesh. Under a particular scenario where clients arrive to the system very close in time, one could think of constructing a tree to deliver the file. One example is *CAN-based Multicast* introduced by Ratnasamy et al. [76]. *CAN-based Multicast* uses an existing scheme on Content-Addressable Networks [75] to construct its tree. Clients are organized in a virtual d-dimensional Cartesian coordinate space. For simplicity, we assume a two-dimensional space, i.e. $d = 2$. In this case, each client is identified by two coordinates (x, y). In addition, each client has, on average, 2×2 neighbors, two neighbors along the X-axis and two others along the Y-axis.
When the server wants to share a file, it delivers the file to all its neighbors. Similarly, each client forwards the file to all its neighbors except the one from which it received the file. While this naive method may produce multiple reception of the file, the authors describe additional optimization to the system to ensure that no client receives from more than one neighbor. As a result, the file propagates in the network as in a tree.

Another example is *Scribe* proposed by Rowstron et al. [80] to support event notification services. Based on *Pastry* [79], a look-up and routing protocol, *Scribe* allows to construct a tree in a completely decentralized way. The server and the clients have each an id and a set of neighbors. Each client receives the file from the neighbor whose id is the closest to the server's id: The server delivers the file to clients that are numerically closest to it, which in turn forward the file to their children and so on.

Our Contribution

From the above discussion, we can largely classify existing approaches for file replication into tree and mesh approaches. Previous work by Biersack et al. [18] revealed that tree approaches can be very efficient for replicating a file over a large number of clients. The authors proved that the number of served clients in tree approaches scales exponentially in time. Mesh approaches, on the other hand, provide more flexibility, which make them more suitable for dynamic environments like P2P networks.

Our first contribution in chapter 5 is a comparison between the scaling behavior of both kinds of approaches. Our results show that mesh approaches are not only simple and flexible, but they can also provide as low service time as tree approaches. Our second contribution is a complete set of simulation results where we identify the main parameters that influence the performance of mesh approaches. Our conclusions provide new insights for the design of new architectures.

1.4 Summary of the Thesis Contributions

This thesis comprises four contributions and a general conclusion. We first address how to provide a large scale VoD service in dedicated overlay networks. Our contribution here is a new video distribution architecture that is highly scalable and very cost-effective. Our architecture is inspired from the *Prefix Caching Assisted Periodic Broadcast* framework: We divide each video into two parts, the prefix and the suffix. The prefix is delivered via a closed-loop scheme from the prefix servers while the suffix is delivered by the central suffix server via an open-loop scheme. This combination of open-loop and closed-loop schemes makes the overall system efficient and, at the same time, ensures an immediate playout of the video.
We then develop a complete cost model, called PS-model, to compute the delivery cost of the video. In the delivery cost, we account not only for the network bandwidth cost but also for the server cost consumed for storing and scheduling the video at the different servers. Our cost formulas assume that the network distribution is a tree with an outdegree m and l levels. The suffix server is placed at the root of the tree while the prefix servers can be placed at any level in the network. For a video whose popularity is known, the PS-model permits to capture the optimal length of the prefix and the level of placement of the prefix servers in the hierarchy. We evaluate our architecture over a broad range of video popularity and system parameters, e.g., the bandwidth to storage cost ratio or the link characteristics. A key result is that our architecture ensures efficiency and scalability.

The second contribution of this thesis is a set of extensions to the PS architecture. For example, we use the PS-model to evaluate the increase in the system cost when the prefix servers can be placed only at fixed levels in the network. We also study how our architecture behaves in the case of short videos, e.g. video clips or clips for product promotions. One important question here would be whether it is worth to divide short videos into prefix and suffix. Another interesting scenario is the dimensioning of a VoD system from scratch; given a set of videos whose popularities are known, we execute the PS-model to find the system resources that we need in terms of storage and I/O. Once the system resources are chosen, we address how the PS architecture adapts to the change in the number and popularity of the videos provided that no modification can be done in the system resources. The last extension is two architectural choices, P-hybrid and S-sat. The first architecture P-hybrid uses a set-top box at the client side to store the prefix of some/all popular videos in the system. Given this option, we derive the gain in the system performance as a function of the local storage capacity of the set-top box. The S-sat architecture, on the other hand, transmits the suffix via satellite rather than the Internet; satellites are very suitable for multicasting data to a large number of clients. We investigate the conditions under which this architecture is efficient.

Our third contribution still deals with VoD but in a P2P environment instead of a dedicated overlay network. The main challenges here are how to efficiently leverage the upload capacity of the clients and how to overcome client departures. Our solution is a new model, called PBA, that ensures simplicity and efficiency at the same time. In contrast to most existing solutions, our pull-based model constructs no multicast trees to deliver the video. Instead, each client performs locally an algorithm to find an available servant from which it downloads the video. We denote by servant an active client that is currently receiving or has already received the video.
We evaluate extensively PBA and compare it to P^2Cast, a multicast tree-based model proposed recently [45]. Our results prove that PBA not only makes the server simple, but it also achieves a good manage-

ment of the system resources. Moreover, we discuss a possible improvement to PBA where we account for the physical distance between clients and their servants.

Our last contribution is a study of the use of P2P networks for file replication services. Existing solutions differ mainly in the way clients organize and cooperate. There are mainly two ways to organize clients in the system, a tree or a mesh. The main advantage of tree approaches is that the number of served clients scales exponentially with time [18]. However, we believe that constructing and maintaining the tree in a dynamic environment such as P2P networks is a very challenging problem. Mesh approaches, on the other hand, are very simple and robust against bandwidth fluctuations and client departures.
Our study here includes two parts. The first part is a comparison between the scaling performance of tree and mesh approaches. Our comparison proves that mesh approaches are not only simple, but they can also outperform tree approaches. The second part of our study is a set of extensive simulation results where we highlight the main parameters that influence the performance of mesh approaches. Our results and conclusions present guidelines for the conception of new architectures depending on the goals to achieve and the environment conditions.

1.5 Road Map of this Thesis

This thesis proceeds as follows. Chapter 2 focuses on VoD in dedicated overlay networks. We introduce our solution, a new video distribution architecture coupled with an analytical cost model, called PS-model. We also present a complete set of results where our architecture appears clearly to achieve efficiency and scalability. In chapter 3 we extend the PS-model to cover many scenarios such as, find the optimal system cost subject to different constraints or the evaluation of different architectural choices. Chapter 4 deals with VoD in P2P networks. There we propose our solution that, in contrast to most of previous ones, ensures an efficient use of the system resources without constructing a multicast tree. In chapter 5 we address the file replication service in P2P networks and advise the use of mesh approaches rather than tree ones. Chapter 6 closes this thesis with a brief summary.

Chapter 2

Cost-optimal Dimensioning of a Large Scale Video on Demand System

2.1 Introduction

Video on Demand (VoD) has become a leading service for a broad range of applications. VoD applications include home entertainment, news on demand, and distance learning, to name but a few. The major challenge in providing a VoD service lies in handling a large number of demands in a real time network, with a real time interaction. The simplest way to achieve so is to allocate a new media delivery stream to each client request when it arrives. This solution has the desirable properties of immediate service, placing minimal demands on client capabilities, and of being simple to implement. However, the bandwidth-intensive nature (usually larger than 1 Mbps) of high quality digital video and the long live nature of the videos (a few min to a few hours) makes the above solution inefficient, non-scalable, and very expensive. Thus, there is a need to serve multiple clients that request the same video at approximately the same time via one single video stream, that is multicast. The idea of using multicast for VoD services was first introduced by Anderson [13]: Early requests for a video can be made to wait for more requests to arrive, and the entire group is served via one single multicast stream. This is referred to as *Batching*. The *Batching* scheme revealed the main feature to provide efficiency (i.e. the use of multicast) and paved the way to propose more efficient schemes that can be broadly classified into open–loop [12, 90, 10, 33, 53, 70, 71, 52, 68, 69, 19, 47, 54] and closed–loop schemes [48, 20, 37, 34, 35, 41, 92, 74].

In this chapter, we address how to provide an efficient and scalable VoD service to a wide client population. We introduce a new video distribution architecture that is highly scalable, very cost effective and, at the same time, ensures a zero start-up delay. Our architecture combines both, open-loop and closed-loop schemes, which makes it suitable for popular and non-popular videos. Given this architecture, we develop a complete cost model, called PS-model, that allows us to minimize the operational cost of the service.

2.1.1 Our Contribution

Our contribution in this chapter is a new video distribution architecture that we inspire from the *Prefix Caching Assisted Periodic Broadcast* framework [44]. We partition each video into a prefix and a suffix. The suffix is stored at the central server while the prefix is stored at one or more prefix servers. A client that wants to view a video joins an already on-going open-loop multicast distribution of the suffix while immediately requesting the prefix of the video as a patch that is sent via *Controlled Multicast* [37]. The novelty of our architecture is a complete cost model: For a video with a given popularity, our cost model gives us the cost-optimal partitioning into prefix and suffix as well as the placement of the prefix servers in the distribution network.

A similar cost model to ours is the one developed by Nussbaumer et al. [64]. As in our model, the authors assume a tree distribution network with l levels with prefix servers deployed at all levels. However, these prefix servers can only store the entire video. Our cost model is more flexible in the sense that any portion of the video can be stored at the prefix servers. In addition, their system cost is simply the sum over the storage cost, the I/O bandwidth cost[1] of the servers, and the network bandwidth cost. Moreover, the cost formulas developed are for simple scenarios with only unicast server/clients connections.

A recent paper by Zhao et al. [97] looks at different network topologies, such as fan-out K, daisy-chain, balanced tree, and network topologies obtained with topology generators. For each of these topologies, they derive tight bounds on the minimum network bandwidth required to serve N clients provided an instantaneous delivery of the video.
There also exist cost models that estimate the I/O or the network bandwidth costs of the system subject to limited storage and bandwidth capacities at the prefix servers [74, 32, 91, 11]. For instance, the model in [32] assumes a central server and many prefix servers with limited I/O bandwidth and storage capacities. Here only a pre-specified fraction of the video, or the full video, can be stored at the prefix servers. Ramesh et al. [74] relax this constraint and allow the prefix servers to store any arbitrary fraction of the video.

In contrast to previous work, in our cost model we

- Model the network as a tree with an outdegree m and l levels.
- Account in the network transmission cost for the number of clients that are simultaneously served by the multicast distribution (either from the prefix servers or the suffix server).
- Allow the prefix servers to be placed at any level in the distribution tree and not only at the last hop between the clients and the network.
- Include in our cost model not only the network transmission cost but also the server cost, which depends on both, the storage occupied and the number of Input/Output (I/O) streams needed.

While the network transmission is a major cost factor, the server cost must be included in the overall cost model, especially when we try to design a cost-optimal video distribution architecture. Otherwise, independent of the popularity of a video, the obvious/trivial architecture will be the one where a large number of prefix servers are placed near the clients. While previous papers have treated in their model the storage space of the prefix servers as a scarce resource, we feel that the cost model can be made more realistic by explicitly modeling the cost of the prefix servers.

Note that in the cost model we assume that videos are constant bit rate encoded (CBR). One could take advantage of previous work [67, 81, 59, 96, 57] to extend our analysis to the case of variable bit rate videos (VBR). However, this is out of the scope of this thesis.

2.1.2 Organization of this Chapter

The rest of this chapter is structured as follows. In section 2.2, we present the transmission algorithms and the content distribution overlay that we use to build the video distribution service. In section 2.3 we introduce our PS-model. In section 2.4, we apply the PS-model to evaluate the performance of our architecture. We give results for different assumptions on the system parameters such as heterogeneous

[1]In this chapter, the I/O bandwidth cost at a given server denotes the bandwidth consumed at that server for stream delivery.

or homogeneous bandwidth costs, zero servers costs, etc. Finally, we conclude this chapter in section 2.5.

2.2 The System Environment

2.2.1 Prefix Caching Assisted Periodic Broadcast

Prefix Caching Assisted Periodic Broadcast[2] [44] assumes that clients are serviced by a main central suffix server and also by local prefix servers, which can be located throughout the network. A video is partitioned into two parts, the prefix and the suffix, which can be of arbitrary proportions. We denote by L (min) the length of the video and D (min) the length of the prefix. Hence, the suffix will have the length $L - D$ (min). The entirety of the prefix is always *viewed before* the suffix. The main idea of the broadcast scheme is that prefix and suffix transmissions should be decoupled in order to transmit each most effectively. The reason why the prefix and suffix are transmitted differently is that the client must receive the prefix *immediately upon request* while the suffix need not be received until the prefix has been completely viewed.

Because the prefix must be immediately received, there is less flexibility in the choice of a transmission scheme for the prefix. In addition, transmitting the prefix from the central server to each client may be costly. In order to reduce transmission costs, the prefix can be stored locally at multiple prefix servers, which can more cheaply transmit the prefix to their local audiences. For the suffix, on the other hand, there is more leeway in the method of broadcast since it need not be received immediately. The allowable delay D in transmitting the suffix permits to deliver it with an open-loop scheme. Therefore, the suffix is retained at the central server, benefiting from sharing amongst a relatively larger number of clients as well as avoiding the server costs incurred to replicate data across multiple servers. Once specific transmission

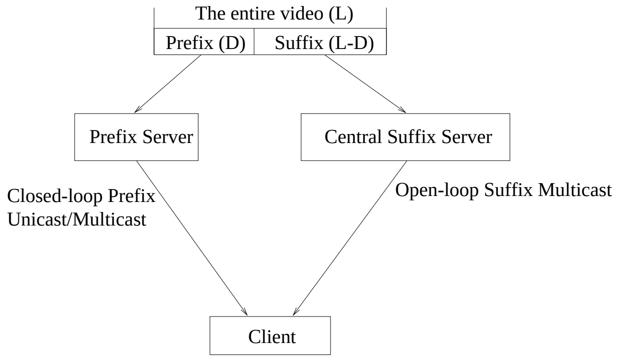


Figure 2.1: The VoD distribution architecture.

schemes for the prefix and the suffix have been chosen, the remaining design parameters are the length of the prefix (and suffix) and the height of placement of the prefix servers in the network. The prefix length and the location of the prefix servers should be chosen so as to efficiently divide the workload between the central suffix server and the prefix servers. In our *Prefix Caching Assisted Periodic Broadcast* model, we choose to transmit the prefix via *Controlled Multicast,* while the suffix is delivered through *Tailored Periodic Broadcast*.

[2]The term broadcast is commonly used in the literature. In our model, broadcast refers to multicast where the data are sent only over links that reach clients that are interested in receiving the video.

2.2.2 The Distribution Network

We assume that the distribution network is organized as an *overlay* network. An overlay network consists of a collection of nodes placed at strategic locations in existing network, i.e. the Internet. Overlay networks provide the necessary flexibility to realize enhanced services such as multicast [51] or content distribution [5] and are typically organized in a hierarchical manner.

In our model, we assume that the topology of our distribution network is a m-ary tree with l levels (figure 2.2). The suffix server is assumed to be at the root. The prefix servers may be placed at any level of the distribution network other than the highest level (i.e. the leaves). If the prefix servers are placed at level j, there will be one at each node of that level, which makes a total of m^j prefix servers. Clients are lumped together at the m^l leaf nodes. The number of clients watching simultaneously a video is not limited to m^l since a leaf node does not represent a single client but multiple clients that are in the same building.

We digress briefly to consider the practical aspects of a network tree model. A tree model captures the hierarchical structure of a large-scale network, where large backbone routers service many smaller service providers which in turn serve the end-user clients. For example, a tree might include multiple levels, dividing the network into national, regional, and local sub-networks. The distribution network

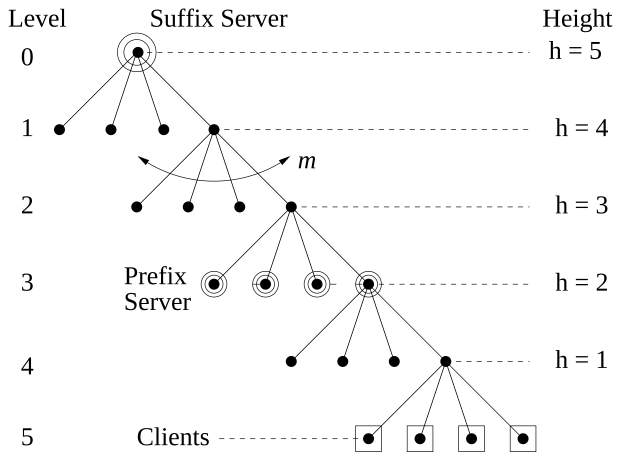


Figure 2.2: Video distribution network.

is assumed to support both unicast and multicast transmissions. Unicast transmission occurs between a server and a single client, whereas multicast transmission occurs when multiple clients (possibly from different leaf nodes) all simultaneously receive the same single transmission from a server. We assume that for the duration of a transmission, a cost must be paid for every link spanned between the server and its active client(s). The per-link cost may differ depending upon the specific links that are utilized.

For a multicast transmission, the cost may change over the duration of the transmission as users join and leave. Note also that if multiple clients reside at a single leaf node then the cost of multicast transmission is effectively the same as if there were only a single client at that node. For clients at different nodes, multicast still offers savings due to links shared at the lower levels by different nodes.

2.2.3 Prefi x Transmission via Controlled Multicast

Patching was first proposed in [48] and then extended with the inclusion of a thresholding policy to produce *Controlled Multicast* [37]. The key idea of patching is to allow clients to share segments of a video stream when they arrive at different times. As the number of clients increases from one to several, the transmission stream is changed from a unicast stream to a multicast one so that late arrivals can still share in the remainder of the stream. In addition, a separate unicast stream must also be transmitted to each client after the first one in order to deliver the data missed due to its later arrival.

For extremely late arrivals, the cost of the additional unicast transmission may outweigh the benefits of sharing in the remaining transmission. *Controlled Multicast* modifies patching to allow for this scenario. Whenever a new transmission is started at time t, arriving clients are patched onto the same stream until time $t + T$, where T is a **thresholding** parameter. The first client to arrive after time $t + T$ is given a brand new transmission, and all future arrivals are patched onto the new transmission instead of the old one, until the threshold time passes again and the process is repeated. Figure 2.3 illustrates the operation of *Controlled Multicast*. At time t_1 the first client arrives: The prefix server starts transmitting the prefix via unicast. When the second client joins at time t_2, the remaining part of the prefix is multicast to clients 1 and 2. Client 2 additionally receives the initial part of the prefix that had been transmitted between t_1 and t_2 via a separate unicast transmission. Since client 3 arrives at time t_3, with $t_3 - t_1 > T$, the prefix server starts a new unicast transmission of the entire prefix.

The costs of *Controlled Multicast* have been shown to increase sub-linearly with the arrival rate of requests and the length of the prefix [74]; however, the analysis assumes a network of a single link between the server and all its clients. This is the case when the prefix servers are located one level above the clients. We refer to this case as **leaf prefix server placement**. Placing the prefix servers higher up in the distribution network increases the cost of transmission; however, it consolidates the arrivals to a smaller number of prefix servers and thereby allows for more sharing to occur amongst clients. Furthermore, placing the prefix servers higher up in the network reduces server costs since there are fewer copies of the prefix. One contribution of this paper is an analysis of the tradeoffs in prefix servers placement for *Controlled Multicast*.

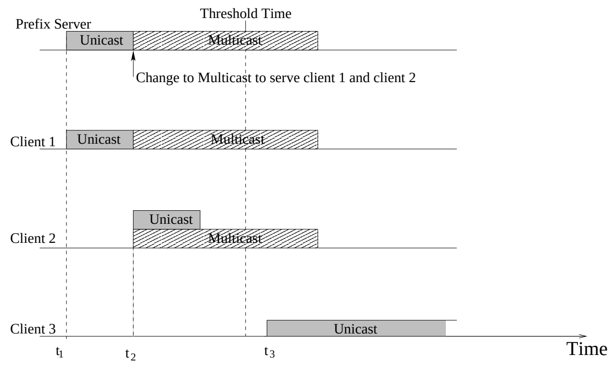


Figure 2.3: Prefix transmission with multicast and patching.

2.2.4 Tailored Periodic Broadcast of the Suffix

We use *Tailored Periodic Broadcast* [19] to transmit the suffix. Thereby, we divide the suffix into segments of fixed lengths. If there are no clients then the suffix server does not transmit. As long as there is at least one client, each segment is periodically multicast at its own transmission rate. Arriving clients receive the multicast of all segments simultaneously. Clients are not expected to arrive at the starting point of each segment; instead, they begin recording at whatever point they arrive, store the data and reconstruct each segment as they receive the data.

The length and rate of each segment is chosen so as to minimize the bandwidth subject to the constraint that each segment must be completely received before its time of playback, and also subject to the storage and reception capabilities of the clients. Figure 2.4 illustrates the *Tailored Periodic Broadcast* scheme for the case of minimal transmission rates. The client starts receiving all segments at time t_0. The shaded areas for each segment contain exactly the content of that segment as received by the client that started recording at time t_0.

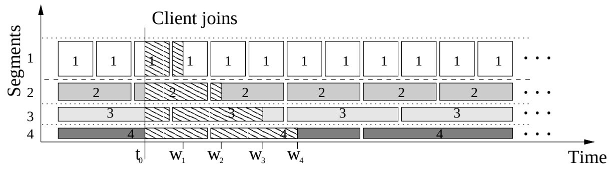


Figure 2.4: The *Tailored Periodic Broadcast* transmission for the case of minimal transmission rate.

Under these assumptions, the total server transmission bandwidth is:

$$R_t^{min} = \sum_{i=1}^{N_s} r_i^{min}$$

where $N_s \triangleq \lceil \frac{L-D}{D} \rceil$ is the number of segments the suffix consists of and r_i^{min} is the minimal transmission rate of segment i. This above figure considers that each segment is broadcast on a separated channel, which requires complex synchronization. In this chapter, we further simplify the Tailored scheme and assume that all segments are transmitted on one single multicast channel at rate R_t^{min}. The server uses time-division multiplexing to ensure that each segment is delivered just in time. When we break the suffix of length $(L-D)$ into N_s segments, each of the segments 1 to (N_s-1) will have a length of D and the last segment N_s has a length of $[(L-D)-(N_s-1)D] = (L-N_sD)$. Segment i with length D needs to be entirely received no later than $iD \cdot 60$ seconds after the beginning of the video consumption. The factor 60 in $iD \cdot 60$ is due to the fact that the lengths of the video, the prefix, and the suffix are expressed in min. Therefore, the minimal transmission rate for segment i is $r_i^{min} = \frac{b \cdot D \cdot 60}{iD \cdot 60} = \frac{b}{i}$, where b (in Mbps) is the consumption rate of the video and $bD \cdot 60$ is the amount of data (in Mb) in a segment of length D.

In the case where the length of the last segment N_s is less than D (i.e. $\frac{L-D}{D}$ is not an integer), the transmission rate $r_{N_s}^{min}$ is computed as follows depending on whether N_s is equal to or larger than 1:

- If $N_s = 1$, the suffix consists of one segment of length $(L-D) < D$ that must be entirely received $D \cdot 60$ seconds after the beginning of the video consumption. The transmission rate of segment N_s then becomes $r_{N_s}^{min} = r_1^{min} = \frac{b \cdot (L-D) \cdot 60}{D \cdot 60} = b\,\frac{L-D}{D}$.

- If $N_s > 1$, the segment N_s should be entirely received $N_sD \cdot 60$ seconds after the beginning of the video consumption. Intuitively, its transmission rate should be set to $r_{N_s}^{min} = \frac{b \cdot (L-N_sD) \cdot 60}{N_sD \cdot 60} = b\,\frac{L-N_sD}{N_sD}$, which is the minimal transmission rate for the case where each segment is broadcast on a separated channel. As we already mentioned, we assume that all N_s segments are multiplexed onto the same multicast channel. Under this assumption, each client remains tuned into that channel until the last segment has been received. As compared to the separated channels case, having one single channel for all segments increases the tuning time of the clients. A larger tuning time means that segments are transmitted during a longer period of time, which wastes network bandwidth in case the video is not popular. In fact, when the video is not popular, a longer transmission of the suffix causes useless transmission of some of the first segments. The reason is that these first segments are transmitted more frequently while there are no interested clients. To prevent useless transmission, we should reduce the tuning time of clients when it is possible. This can be done by increasing the transmission rate of segment N_s and transmit it ones each $(N_s - 1)D \cdot 60$ seconds instead of $N_s D \cdot 60$ seconds. For a video with a low demand, the increase in the transmission rate r_{N_s} of the last segment is set-off by avoiding useless transmissions of some of the first segments. For a popular or very popular video, the prefix length is quite short, which means that $N_s \sim (N_s - 1)$ and consequently, the growth in r_{N_s} has a negligible influence on the overall server transmission bandwidth R_t^{min}. From the above analysis and using the fact that $N_s = \lceil \frac{L-D}{D} \rceil = \lfloor \frac{L}{D} \rfloor$, the transmission rate of segment N_s then becomes $r_{N_s}^{min} = \frac{b \cdot (L - N_s D) \cdot 60}{(N_s - 1) D \cdot 60} = b \, \frac{(\frac{L}{D} - \lfloor \frac{L}{D} \rfloor) D}{(N_s - 1) D} = b \, \frac{\frac{L}{D} - \lfloor \frac{L}{D} \rfloor}{N_s - 1}$.

The total server transmission bandwidth is therefore:

$$R_t^{min} = \begin{cases} b\,(\displaystyle\sum_{i=1}^{\lfloor \frac{L-D}{D} \rfloor} \frac{1}{i} + \frac{\frac{L}{D} - \lfloor \frac{L}{D} \rfloor}{N_s - 1}) & \text{if } N_s > 1 \\ b\, \dfrac{L - D}{D} & \text{if } N_s = 1 \end{cases}$$

When a new client arrives, it tunes to the multicast channel and starts recording the data. As a consequence, parts of the video will be received some time before they are consumed and must be stored by the client in the meantime. Figure 2.5 plots the evolution with time of the amount of data stored for a video of length $L = 90$ min and a prefix length of $D = 2$ min. We see that at the beginning, more and more data will be received ahead of time so that the amount of data keeps increasing until it reaches a peak corresponding to nearly 40 percent of the video. From there on, data will be consumed at a rate higher than the aggregate rate at which new data are received and the storage curve decreases. Storing up to 40 percent of the video data does not pose any problem with todays equipment. In fact, there already exist products such as the digital video recorder by TiVo [6] that can store up to 60 hours of MPEG II encoded video.

The *Tailored Periodic Broadcast* scheme also allows to support user interactions [17] such as fast forward if the transmission rate of each segment is increased by a small factor (less than twice the minimal transmission rate), provided that the client has enough buffer to store large parts of the video.

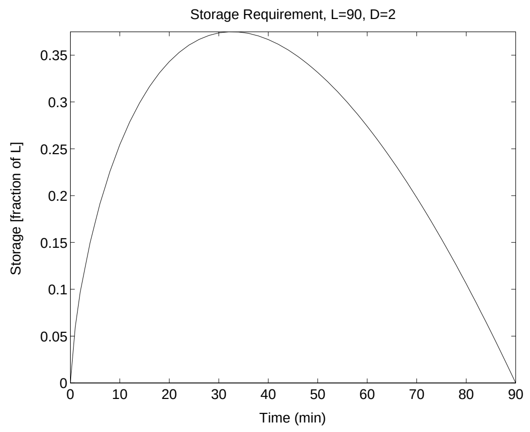


Figure 2.5: Storage requirement at the clients as a function of time for $L = 90$ min and $D = 2$ min.

2.2.5 Interaction Between Clients and Servers

When a new client wants to receive a video, it sends a request to its responsible prefix server in the distribution tree. We assume that each client knows its responsible prefix server, which can be the closest one in terms of physical distance or in terms of delay. In the later case, the responsible prefix server of a client may change dynamically as the network conditions change. Upon receiving the request, the prefix server will either start a new transmission cycle of the prefix or extend the ongoing prefix multicast transmission for the new client and transmit a unicast patch to that client. However, the client does not need to contact the suffix server. The suffix server simply needs to know whether there is currently at least one client that needs to receive the suffix, which the suffix server can learn by communicating with the prefix servers. Other large scale video distribution schemes such as the *Hierarchical Multicast Stream Merging* scheme [35] require that all client requests be handled by the central server, which makes the central server a potential bottleneck. Our video distribution scheme is not only *scalable* in terms of the network and server resources required to stream a video to a large number of clients but also in terms of processing incoming client requests.

2.3 PS-model, a Cost Model for the Video Transmission

2.3.1 Introduction

We divide the costs of a VoD system into network and server costs. The network costs are proportional to the amount of network bandwidth which is spanned over each link between a server and its clients. The server cost is dependent upon the necessary storage and upon the total number of I/O streams which the server(s) must simultaneously support over the network. It is interesting to note that the storage and I/O stream capacity of a server cannot be purchased in arbitrary quantities. Usually, they can only be purchased in discrete increments. As a result, it is unlikely that a server can exactly match both storage and streaming requirements; typically one constraint will be slack as the server will either have extra storage or extra streaming capability.

We will examine the expected delivery cost for a video of length L as a function of the average request rate per min λ, the prefix length (and allowable suffix delay) D, and the topology of the network. The storage and I/O capacities should be fixed for each server; however, to facilitate the analysis we will assume that any number of streams can be allocated with the costs paid on a per-stream basis.

We divide the multicast tree into levels $1, \ldots, l$, where level 1 consists of the m links from the root and level l consists of the m^l links connected to the leaf nodes. Arrivals to a link at level j are modeled as a Poisson process with parameter λ/m^j. As a consequence, arrivals at the root will form a Poisson process with parameter λ. We will derive the cost for the prefix and for the suffix transmission separately and then combine both to obtain the overall system cost. The costs for the prefix are first derived for a single prefix server at the root. We next generalize the root case to the general case where the prefix servers are placed at some level between the root and the clients.

We neglect the effects of network latency, even though they can be important from an operational point of view. One might treat latency effects by constraining the maximum number of hops allowed between prefix servers and their clients. We will refer to this cost model as the **PS-model** to distinguish it from other cost models.

2.3.2 Costs for Prefi x Transmission

Prefix Server at the Root

The costs for the prefix fall into three categories: network, storage capacity and I/O capacity. We first consider a single prefix server at the root of the multicast tree. We will afterwards generalize our results to the case where the prefix server is at an arbitrary height in the tree.

Network Bandwidth Costs for Prefix Server Placed at the Root A single server at the root combines many small request arrival streams (to the leaves) into a single large arrival stream (to the root); by combining all the requests, we increase the possibility for sharing with *Controlled Multicast*. However, this is counterbalanced by the fact that sharing is no longer free; if two clients share the same I/O stream but reside on different leaves, a separate network cost must be paid for each of them. Of course, if clients are already active at every leaf node, then no new network costs will be paid for future arrivals. Yet, this scenario is unlikely to happen even for high arrival rates. Indeed, high arrival rates produce short threshold times in order to reduce the length of the unicast streams.

Let t_i be the time of the ith complete multicast transmission of the prefix without any patching. Arrivals between times t_i and t_{i+1} will share from the multicast transmission at time t_i and will each receive a separate unicast transmission for the data which were missed. We can divide the patching process up into separate renewal cycles $(t_1 t_2], (t_2 t_3], \ldots$ which are independents and identically distributed in their usage of bandwidth. We therefore analyze the bandwidth usage over a single renewal cycle.

Given the threshold time T (min), on average there will be $T\lambda$ arrivals which will each need a partial transmission of the prefix in unicast. The average length of the unicast transfer will be $T/2$ since the arrivals are uniformly distributed over time. Finally a bandwidth cost must be paid for every link on the path between clients (at the leaves) and the root server. As a result, the total amount of data transmitted for the unicast streams over one renewal cycle is

$$C_{netw}^{unicast} = b \cdot 60 \, \frac{lT^2\lambda}{2}.$$

Each arrival will also share from a single multicast network stream. A price must be paid for every link in use. Given a link at level j, let τ_j be the duration of time in which the link is active. For the multicast stream, a link is active from the time of the first arrival (before time T) to that link to the end of the prefix. Arrivals to a link at level j form a Poisson process with parameter λ/m^j. As each renewal cycle begins with the activation of a stream between the root and a single client, we know that one link at each level will be active at time zero. Therefore $\tau_j = D$ with probability $1/m^j$. We will now write out an expression for $\mathbb{E}[\tau_j]$:

$$\begin{aligned}\mathbb{E}[\tau_j] &= D\mathbb{P}\{\tau_j = D\} + \mathbb{E}[\tau_j | \tau_j \neq D]\mathbb{P}\{\tau_j \neq D\} \\ &= D\frac{1}{m^j} + \mathbb{E}[\tau_j | \tau_j \neq D]\frac{m^j - 1}{m^j}.\end{aligned}$$

Given a Poisson process with parameter λ/m^j, the time of first arrival will have an exponential distribution with parameter λ/m^j and a cumulative distribution $F(t) = 1 - e^{-\frac{\lambda}{m^j}t}$. We evaluate $\mathbb{E}[\tau_j | \tau_j \neq D]$

making use of the fact that $\int_0^T t\frac{\lambda}{m^j}e^{-\frac{\lambda}{m^j}t}dt = \int_0^T (e^{-\frac{\lambda}{m^j}t} - e^{-\frac{\lambda}{m^j}T})dt$.

$$\begin{aligned}
\mathbb{E}[\tau_j|\tau_j \neq D] &= \int_0^T (D-t)\frac{\lambda}{m^j}e^{-\frac{\lambda}{m^j}t}dt \\
&= \int_0^T D\frac{\lambda}{m^j}e^{-\frac{\lambda}{m^j}t}dt - \int_0^T t\frac{\lambda}{m^j}e^{-\frac{\lambda}{m^j}t}dt \\
&= D(1-e^{-\frac{\lambda}{m^j}T}) - \int_0^T (e^{-\frac{\lambda}{m^j}t} - e^{-\frac{\lambda}{m^j}T})dt \\
&= D(1-e^{-\frac{\lambda}{m^j}T}) - (-\frac{m^j}{\lambda}e^{-\frac{\lambda}{m^j}t} - e^{-\frac{\lambda}{m^j}T}t)\Big|_{t=0}^{T} \\
&= D(1-e^{-\frac{\lambda}{m^j}T}) - \frac{m^j}{\lambda} + \frac{m^j}{\lambda}e^{-\frac{\lambda}{m^j}T} + Te^{-\frac{\lambda}{m^j}T}.
\end{aligned}$$

Substituting $\mathbb{E}[\tau_j|\tau_j \neq D]$ into $\mathbb{E}[\tau_j]$ produces

$$\begin{aligned}
\mathbb{E}[\tau_j] &= D\frac{1}{m^j} + \mathbb{E}[\tau_j|\tau_j \neq D]\frac{m^j-1}{m^j} \\
&= D\frac{1}{m^j} - \left(\frac{m^j}{\lambda} - \frac{m^j}{\lambda}e^{-\frac{\lambda}{m^j}T} - Te^{-\frac{\lambda}{m^j}T} - D(1-e^{-\frac{\lambda}{m^j}T})\right)\frac{m^j-1}{m^j} \\
&= D\left(\frac{1}{m^j} + (1-e^{-\frac{\lambda}{m^j}T})\frac{m^j-1}{m^j}\right) - (1-e^{-\frac{\lambda}{m^j}T})\frac{m^j-1}{\lambda} + (\frac{m^j-1}{m^j})Te^{-\frac{\lambda}{m^j}T} \\
&= D(1-e^{-\frac{\lambda}{m^j}T}(1-\frac{1}{m^j})) - (1-e^{-\frac{\lambda}{m^j}T})\frac{m^j-1}{\lambda} + (\frac{m^j-1}{m^j})Te^{-\frac{\lambda}{m^j}T}.
\end{aligned}$$

We find the total multicast cost $C_{netw}^{multicast}$ by summing over all the links in the tree:

$$C_{netw}^{multicast} = b \cdot 60 \sum_{j=1}^{l} m^j \mathbb{E}[\tau_j]$$

and the average network bandwidth cost C_{netw}^{root} can be found as the sum of $C_{netw}^{unicast}$ and $C_{netw}^{multicast}$ divided by the average duration of each renewal cycle $(T+1/\lambda)\cdot 60$.

$$\begin{aligned}
C_{netw}^{root} &= \frac{C_{netw}^{multicast} + C_{netw}^{unicast}}{(T+1/\lambda)\cdot 60} \\
&= b\,\frac{\sum_{j=1}^{l} m^j \mathbb{E}[\tau_j] + lT^2\lambda/2}{T+1/\lambda}.
\end{aligned}$$

Server Costs for Prefix Server Placed at the Root It is easy to see that the storage cost C_{sto}^{root} of a single root server will be $b \cdot 60D$. The I/O stream cost must be paid for the output capabilities of each server, i.e. the number of I/O streams which a server can simultaneously maintain. From the expression of C_{netw}^{root} above, one can conclude that the average number of streams for a root server is equivalent to

$$C_{I/O}^{root} = b\,\frac{D + T^2\lambda/2}{T+1/\lambda}.$$

If T is chosen so as to minimize the number of I/O streams, then $C_{I/O}^{root} = b(\sqrt{2D\lambda+1}-1)$. However, choosing T to minimize the number of concurrent I/O streams will unnecessarily minimize the network bandwidth. If T is chosen to minimize the network bandwidth, then

$$C_{I/O}^{root} = b(\sqrt{2CD\lambda/l + 1} - 1 - \frac{D\lambda(C/l - 1)}{\sqrt{2CD\lambda/l + 1}}),$$

where C is a scalar constant. In case of a non-popular video that has on average, at most, one arrival in an interval of time D ($\lambda < 1/D$), each request activates a new prefix transmission and then a new cycle. Hence, the threshold time T becomes $zero$ and the expressions for C_{netw}^{root} and $C_{I/O}^{root}$ simplify to

$$\begin{aligned} C_{netw}^{root} &= bl\lambda D \\ C_{I/O}^{root} &= b\lambda D\,. \end{aligned}$$

Varying the Placement of the Prefix Server

We now generalize the model to the case where the prefix servers can be placed at any height in the network tree. By placing the prefix servers at some height h in the tree where $1 < h < l$, we divide the arrival process between m^{l-h} servers, each of which can be considered as the root of a network tree with height h. We need only therefore to consider the root server case for a tree of the proper height h with arrival rate λ/m^{l-h}, and then multiply the costs by the number of servers m^{l-h}. The resulting formulas are listed in table 2.1. The height $h = 1$ refers to the case of a **leaf server**, i.e. one level before the clients (see figure 2.2).

Cost terms	
C_{netw}^{prefix}	$b \cdot m^{l-h}(\frac{\frac{hT^2\lambda}{2m^{l-h}} + \sum_{j=1}^{h} m^j \mathbb{E}[\tau_j]}{T + m^{l-h}/\lambda})$ $[\mathbb{E}[\tau_j] = D(1 - e^{-\frac{\lambda}{m^j m^{l-h}}T}(1 - \frac{1}{m^j})) - (1 - e^{-\frac{\lambda}{m^j m^{l-h}}T})\frac{m^{l-h}(m^j - 1)}{\lambda} + (\frac{m^j - 1}{m^j})Te^{-\frac{\lambda}{m^j m^{l-h}}T}]$
C_{sto}^{prefix}	$b \cdot 60 \cdot m^{l-h} \cdot D$
$C_{I/O}^{prefix}$	$b \cdot m^{l-h} \cdot (\frac{D + \frac{T^2\lambda}{2 \cdot m^{l-h}}}{T + m^{l-h}/\lambda})$

Table 2.1: Summary of the prefix cost terms for the PS-model when the prefix servers are placed at height h, i.e. level $(l - h)$. The tree distribution has an outdegree m and comprises l levels.

2.3.3 Costs for Suffi x Transmission with a Multicast Tree

The costs once again fall into three categories: bandwidth, storage capacity and streaming capacity.
The bandwidth cost is equal to the transmission rate R_t^{min} multiplied by the average number of active links, which we will now calculate. For the *Tailored Periodic Broadcast*, each arrival is serviced for an amount of time $\mathbb{E}[S_t]$ (min), which equals the transmission time of the last segment:

$$\mathbb{E}[S_t] = \begin{cases} (N_s - 1) \cdot D = \lfloor \frac{L-D}{D} \rfloor \cdot D & \text{if } N_s > 1 \\ D & \text{if } N_s = 1 \end{cases}$$

As we have already mentioned, we assume that all segments are multiplexed onto one single multicast channel. As a consequence, each client will consume a bandwidth of R_t^{min} during all the transmission of the suffix. If one multicast channel were dedicated to each segment, the bandwidth consumption could be reduced; the client would be connected only to channels corresponding to segments not yet received. However, this reduction in bandwidth cost comes at the expense of a more complex multicast transmission and a complex synchronization between channels. This study is left for future work. From queuing theory, it can be shown that given an expected service time $\mathbb{E}[S_t]$ and memoryless arrivals with parameter $\frac{\lambda}{m^j}$, the probability of n requests simultaneously in progress is given by

$$\mathbb{P}\{n \text{ requests}\} = \frac{e^{-\frac{\lambda}{m^j}\mathbb{E}[S_t]}(\frac{\lambda}{m^j}\mathbb{E}[S_t])^n}{n!},$$

which is a Poisson distribution. This result can be found through the derivation of the Erlang call-blocking formula commonly used in telecommunications. Arrivals to a link at level j are memoryless with parameter λ/m^j. Define $P(j)$ as the probability that a link at level j has at least one request:

$$P(j) \triangleq 1 - \mathbb{P}\{0 \text{ requests}\} = 1 - e^{-\frac{\lambda}{m^j}\mathbb{E}[S_t]}$$

Then

$$P(j) = \begin{cases} 1 - e^{-\frac{\lambda \lfloor \frac{L-D}{D} \rfloor D}{m^j}} & \text{if } N_s > 1 \\ 1 - e^{-\frac{\lambda D}{m^j}} & \text{if } N_s = 1 \end{cases}$$

The expected number of active links at any given time is therefore

$$\mathbb{E}[\text{active links}] = \sum_{j=1}^{l} m^j P(j),$$

and the bandwidth is

$$C_{netw}^{suffix} = R_t^{min} \sum_{j=1}^{l} m^j P(j). \tag{2.1}$$

On the other hand, the suffix is continuously and periodically multicast independent of the arrival rate as long as there is at least one active client in the network (if there are no clients at all, the central server will not send the suffix). The I/O stream cost is therefore equal to $R_t^{min} \times P(0)$, where $P(0)$ is the probability that there is at least one request in progress at the root (i.e. one active client in the system):

$$C_{I/O}^{suffix} = \begin{cases} b\,(\sum_{i=1}^{\lfloor \frac{L-D}{D} \rfloor} \frac{1}{i} + \frac{\frac{L}{D} - \lfloor \frac{L}{D} \rfloor}{N_s - 1})(1 - e^{-\lambda \lfloor \frac{L-D}{D} \rfloor D}) & \text{if } N_s > 1 \\ b\,\frac{L-D}{D}(1 - e^{-\lambda D}) & \text{if } N_s = 1 \end{cases}$$

The storage cost is proportional to the length of the suffix, which is $C_{sto}^{suffix} = b \cdot 60(L - D)$. The terms of the suffix costs are given in table 2.2, while table 2.3 shows all the important mathematical terms that we use through this chapter.

Cost terms		
C^{suffix}_{netw}	$b\,(\sum_{i=1}^{\lfloor\frac{L-D}{D}\rfloor}\frac{1}{i}+\frac{\frac{L}{D}-\lfloor\frac{L}{D}\rfloor}{N_s-1})\sum_{j=1}^{l}m^j(1-e^{-\frac{\lambda\lfloor\frac{L-D}{D}\rfloor D}{m^j}})$	if $N_s\ >\ 1$
	$b\,\frac{L-D}{D}\sum_{j=1}^{l}m^j(1-e^{-\frac{\lambda D}{m^j}})$	if $N_s\ =\ 1$
C^{suffix}_{sto}	$b\cdot 60\,(L-D)$	
$C^{suffix}_{I/O}$	$b\,(\sum_{i=1}^{\lfloor\frac{L-D}{D}\rfloor}\frac{1}{i}+\frac{\frac{L}{D}-\lfloor\frac{L}{D}\rfloor}{N_s-1})(1-e^{-\lambda\lfloor\frac{L-D}{D}\rfloor D})$	if $N_s\ >\ 1$
	$b\,\frac{L-D}{D}(1-e^{-\lambda D})$	if $N_s\ =\ 1$

Table 2.2: Summary of the suffix cost terms for the PS-model.

Term	**Definition**
m	Tree breadth
l	Tree depth
h	Prefix server height
L	Video length (min)
D	Prefix length (min)
$L-D$	Suffix length (min)
N_s	Number of segments of the suffix ($N_s \triangleq \lceil\frac{L-D}{D}\rceil$)
λ	Average client request arrival rate (1/min)
N	Video popularity ($N \triangleq \lambda L$)
τ_j	Active occupation duration for link at depth j (prefix transmission)
$P(j)$	Prob. link at depth j is active (suffix transmission)
T	Threshold time (min)
b	Consumption rate of the video [Mbps]
C	Scalar constant

Table 2.3: Important Mathematical Terms used in this chapter.

2.3.4 Overall System Cost for the Case of a Single Video

We divide the costs of the VoD service into network and server costs. The network costs are proportional to the amount of network bandwidth that is expended for the transmission of the prefix and the suffix. The server costs depend upon the necessary storage and upon the total number of I/O streams needed for the suffix server and the prefix server(s). The total cost of the system can be computed as the sum of the total network and total server costs:

$$C_{PS}^{system} = C_{netw}^{system} + \gamma C_{server}^{system} \tag{2.2}$$

To relate the network and the server costs, a normalization factor γ is introduced that allows us to explore various scenarios for the cost of the servers as compared to the cost for the network bandwidth. We consider here the values of $\gamma = \{0, 0.1, 1\}$. The case of $\gamma = 0$ corresponds to the case that only the cost for network transmission is taken into account and the cost for the servers is not considered at all (considered to be zero). $\gamma = 0.1$ provides high network cost relative to the server cost, while $\gamma = 1$ represents the case where the network cost is relatively low as compared to the server cost.

The terms for the network and server costs are given by:

$$\begin{aligned} C_{netw}^{system} &= C_{netw}^{prefix} + C_{netw}^{suffix} \\ C_{server}^{system} &= C_{server}^{prefix} + C_{server}^{suffix} \end{aligned}$$

As stated in [16], the server cost depends on both, the required amount of storage C_{sto} (in Mb) and the amount of disk I/O bandwidth $C_{I/O}$ (in Mbps).

$$\begin{aligned} C_{server}^{prefix} &= \max(C_{I/O}^{prefix}, \beta C_{sto}^{prefix}) \\ C_{server}^{suffix} &= \max(C_{I/O}^{suffix}, \beta C_{sto}^{suffix}) \end{aligned}$$

To be able to relate the costs for storage and for I/O, we introduce the normalization factor β that is determined as follows: If our server has a storage capacity of d_{sto} [Mb] and an I/O bandwidth of $d_{I/O}$ [Mbps], then $\beta \triangleq \frac{d_{I/O}}{d_{sto}}$. Since the server will be either I/O limited (I/O is the bottleneck and no more requests can be served) or storage limited (storage volume is the bottleneck and no more data can be stored), the server cost is given as the *maximum* of $C_{I/O}$ and βC_{sto}.

In addition, we introduce the parameter *lhc* to model the case where the cost for the “last-hop link” towards the clients is not the same as the cost for other links. For instance, a value of $lhc = 0.1$ means that the cost for the last link to the clients is ten times cheaper than the cost for the links higher up in the hierarchy.

2.4 Results for Long Videos

2.4.1 Introduction

We consider a distribution network with an out-degree $m = 4$ and a number of levels $l = 5$. We expect that such a topology is representative for a wide distribution system that covers a large geographical areas of the size of a country such as France or the UK. If one wants to model a densely populated metropolitan area such as NewYork, one would choose $l < 5$ (e.g. $l = 2, 3$) and $m > 4$ (e.g. $m = 10$). Our model has quite a few parameters and we present results only for a limited subset of parameter values that can provide new insights. For the rest of the chapter, we will vary only the parameters γ, last-hop cost *lhc*, and video length L. The other parameters are chosen as follows: For the disk I/O cost to disk storage cost ratio β, we choose $\beta = 0.001$, which is a realistic value for the current disk technology such as the IBM Ultrastar 72ZX disk. The video length will be $L = 90$ min for most of the time, except when we consider very short video clips of lengths $L = 2$ and 7 min (section 3.3).

2.4.2 Homogeneous Link Costs

For the first set of results that we present, the server cost is weighted by $\gamma = 1$ and the network per-link costs are uniform at all levels of the network ($lhc = 1$). We first plot in figure 2.6 the optimal system cost as a function of the video popularity N. We define the video popularity N as the average number of clients requesting a same video in an interval of time of duration L ($N \triangleq \lambda L$). In figure 2.6 we see

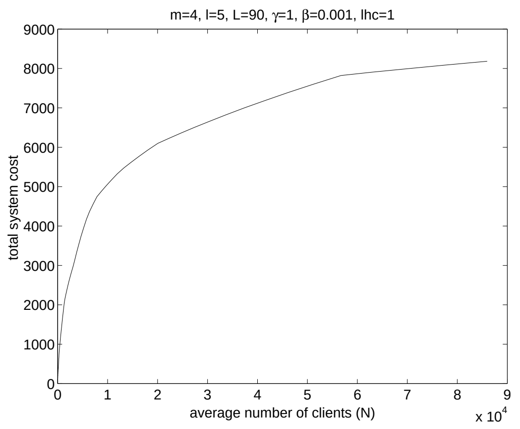
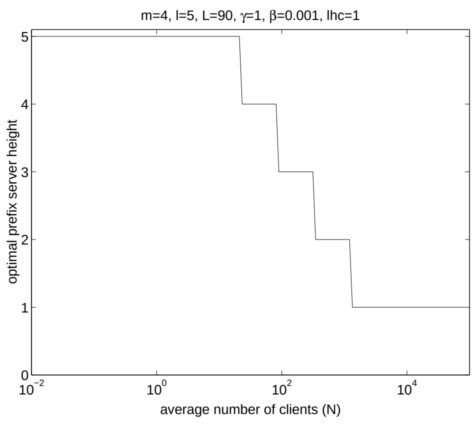


Figure 2.6: Total system cost C^{system} with optimal prefix server height and optimal prefix length, for $\gamma = 1$ and $lhc = 1$.

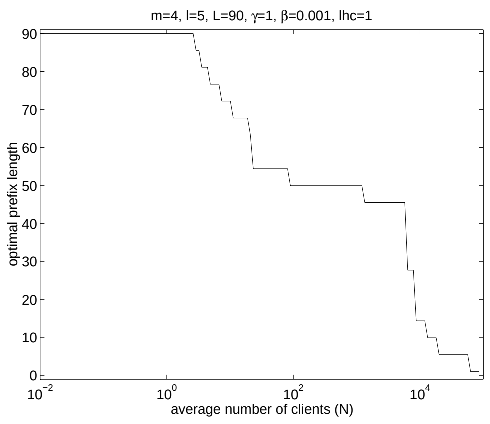
that the total cost efficiency improves with increasing video popularity N: For a 10-fold increase in the value of N from 9,000 to 90,000 clients, the system cost only doubles.

The optimal values for the prefix length (min) and prefix server placement in the hierarchy as a function of the video popularity N are given in figures 2.7(a) and 2.7(b). We see that both, the prefix server height and the prefix length decrease monotonically with N. For videos that are rarely demanded ($N < 20$), the prefix server is placed at the *root* and the optimal prefix comprises the whole video of 90 min. Indeed, for $N < 20$, the storage cost due to a replication of the prefix in multiple prefix servers is not justified and the optimal architecture is a *centralized* one. On the other hand, for videos that are popular or very popular, the optimal architecture is a *distributed* one with the server for the suffix at the root and the prefix servers closer to clients. As the popularity N increases, the optimal prefix length decreases since the transmission bandwidth required by the prefix server increases with the square root of the number of clients served, while for the suffix server, the transmission bandwidth required depends, for very high values of N, only on the length of the suffix and not on the number of clients served. We plot the prefix-suffix cost breakdown in figure 2.8. We see that the suffix cost C^{suffix} is initially lower than the prefix cost C^{prefix} since the prefix length is quite long. Eventually, for an increasing video popularity N, the suffix system becomes more cost efficient, and so the length of the prefix is significantly reduced and the suffix cost becomes greater than the prefix cost. From figure 2.8(c) we see that the suffix cost C^{suffix} for higher values of N is entirely determined by the suffix *network* cost. In the following, we will not present the suffix cost breakdown anymore since it does not provide any additional insight. For a given suffix length, the fact that C^{suffix} does not change with N indicates that *all* the links of the video distribution network are active, i.e. the multicast transmission becomes a broadcast to all leaf nodes.

If we take a closer look at the evolution of the prefix *server* cost for very popular videos with $N > 10^3$ (figure 2.8(b)), we see that the prefix server cost increases linearly with increasing N. In this case, to achieve an optimal system cost C^{system}, the prefix length is frequently shortened.

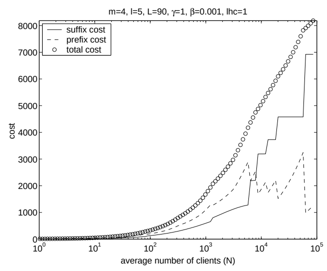


(a) Optimal prefi x server height

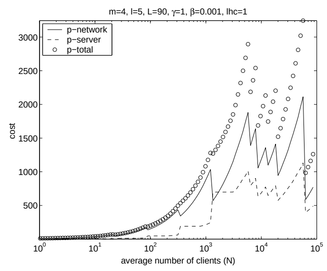


(b) Optimal prefi x length

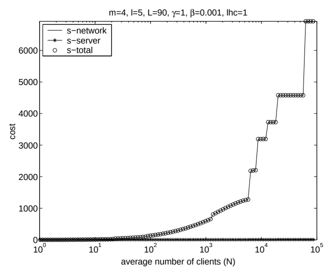
Figure 2.7: Optimal prefix server height and optimal prefix length for $\gamma = 1, lhc = 1$.



(a) total system cost



(b) prefi x cost breakdown



(c) suffi x cost breakdown

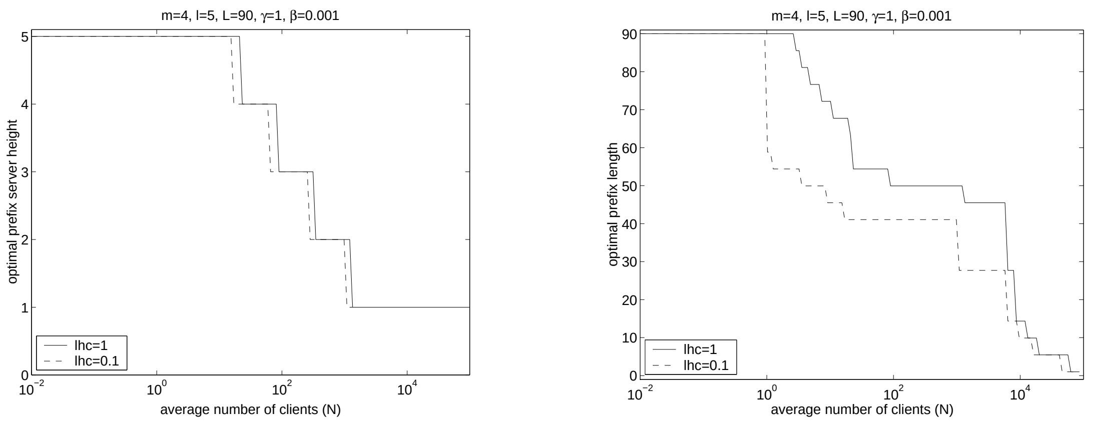
Figure 2.8: Breakdown of costs for $\gamma = 1$ and $lhc = 1$.

2.4.3 Heterogeneous Link Costs

We now set the cost of transmission between levels 4 and 5 (the ”last-hop” from the root to the clients at the leaves) to be one-tenth the normal network cost. A reduced last-hop link cost would apply to situations where the local service providers are much cheaper than their regional and national counterparts since they support less traffic. A recent study [15] contains a cost comparison for transmission links of different bandwidth that indicates that the cost in Mb/hour for local access via ADSL or Cable is at least one order of magnitude lower than the cost for a high speed OC-3 or OC-48 link. We can model this case by using a reduced last-hop link cost, which is $\frac{1}{10}$ of the cost for the other links. We refer to this case as $lhc = 0.1$.

In figure 2.9, we plot the optimal prefix lengths and prefix server heights, with $m = 4, l = 5, \gamma = 1$, and $lhc = \{1, 0.1\}$. Reducing the last-hop cost makes network transmission cheaper compared to server cost. We see that the prefix server heights are roughly the same while the prefix lengths decay faster than in the original setup ($lhc = 1$) to compensate for the relative increase in prefix server costs. This may seem counter intuitive since reducing the last-hop cost should make the prefix cost cheaper, especially if the prefix servers are at the leaves while the suffix server is at the root. However, we see in figure 2.10(b) that the *server* cost for the prefix C^{prefix}_{server} now dominates the total prefix cost C^{prefix}, (in particular when the prefix servers are placed at the leaves, i.e. for $N > 10^3$) which was not the case for $lhc = 1$ (see figure 2.8(b)).

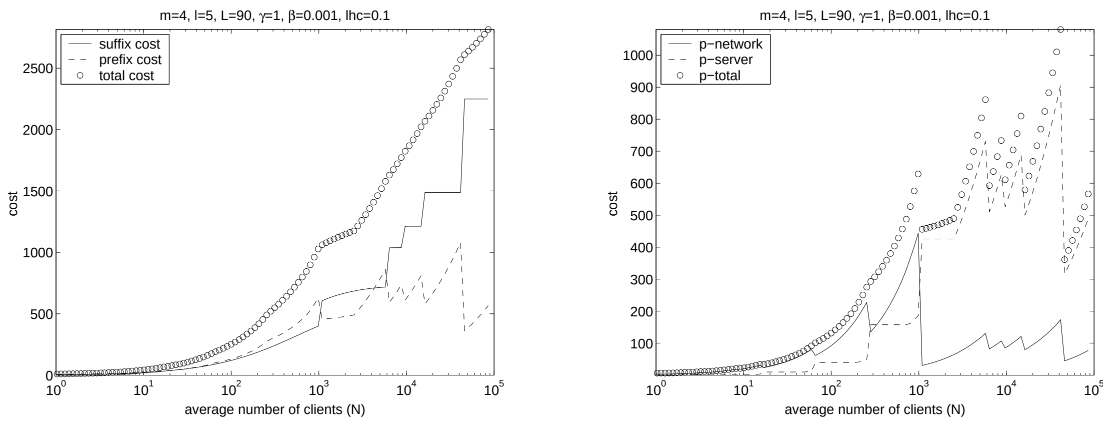
When we compare the suffix costs in figures 2.8 and 2.10, we note that reducing the last-hop cost reduces the suffix network cost by almost a factor of 4, which assures that the suffix system remains cost



(a) optimal prefi x server height (b) optimal prefi x length

Figure 2.9: Optimal prefix server height and optimal prefix length for $\gamma = 1$ and $lhc = \{1, 0.1\}$.

effective. This cost reduction is due to the fact that with $m = 4, l = 5$, there are **1024 links** at the leaf level (last-hop) and only **340 links** in the rest of the network. The network cost for the suffix system and the server cost for the prefix system are roughly in the same neighborhood. As a result, the server cost (i.e. the prefix server cost) is magnified in importance and the optimal policy is to compensate by shortening the prefix.



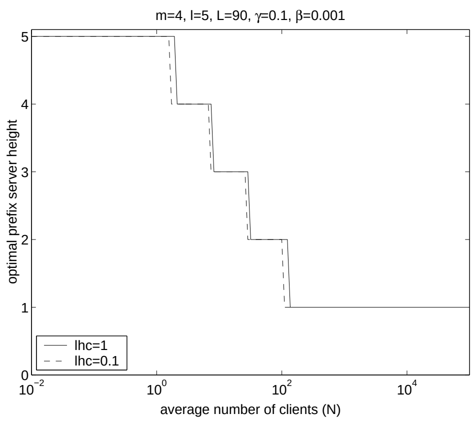
(a) total system cost (b) prefi x cost breakdown

Figure 2.10: Breakdown of costs for $\gamma = 1$ and $lhc = 0.1$.

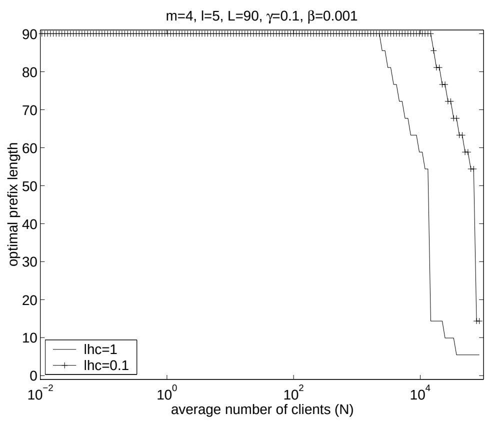
2.4.4 Reducing the Server Costs relative to the Network Transmission Cost

While the costs for servers are usually comparable in Europe and the US, the cost of network transmission is much *higher* in Europe than in the US. To account for the case where the network transmission cost relative to the server cost is high, we set γ to 0.1. The results for $\gamma = 0.1$ (figure 2.11) indicate that reduced relative server cost allows to deploy more servers in order to reduce the impact of the expensive network transmission cost. If we compare with $\gamma = 1$ (figure 2.9), we see that for $\gamma = 0.1$

- The optimal prefix (figure 2.11(b)) comprises the entire video for a much wider range of client population N. Only for a very high video popularity $N > 10^4$, the optimal prefix length decreases significantly.



(a) optimal prefi x server height

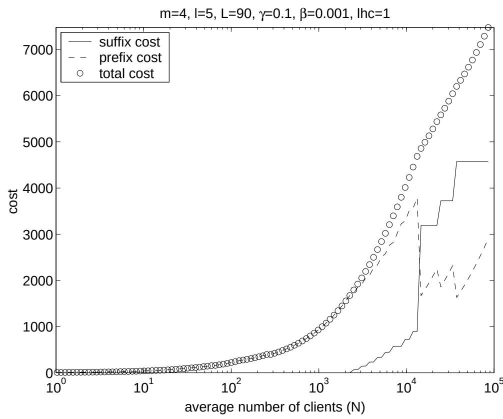


(b) optimal prefi x length

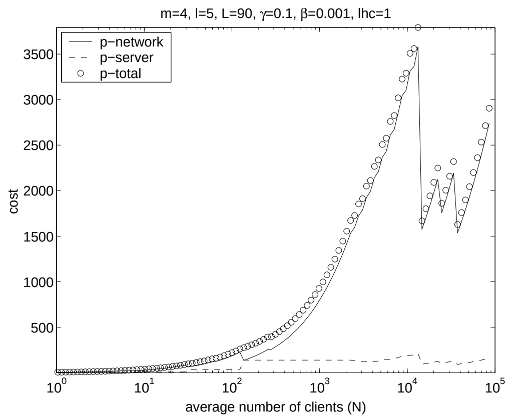
Figure 2.11: Optimal prefix server height and optimal prefix length for $\gamma = 0.1$.

- The prefix server height (figure 2.11(a)) drops faster to $h = 1$ since this helps to reduce the network costs and since the server costs, though they increase (the number of prefix server increases), have been discounted.

We also examine the case where $\gamma = 0.1$ and in addition the network cost for the last-hop is reduced to $lhc = 0.1$. We plot the optimal prefix lengths and prefix server positions in figure 2.11. Previously, we saw for $\gamma = 1, lhc = 0.1$ that although the reduced last-hop cost significantly diminished the cost of prefix service (and of suffix service), the high server costs associated with leaf servers limited the use of the prefix. Now the server cost has been discounted, we see that reducing the last-hop network cost will allow the prefix servers to deliver the entire video over a wider range of video popularities as compared to the $lhc = 1$ case (figure 2.11). This is interesting, as reducing the last-hop network cost *decreased* the prefix length in the $\gamma = 1$ case and is an example of the interplay between network and server costs. For $\gamma = 1, lhc = 0.1$ the prefix server cost dominates the overall prefix cost (see figure 2.10(b), while for $\gamma = 0.1, lhc = 0.1$ it is the prefix network cost that dominates the overall prefix cost (see figure 2.13(b)).

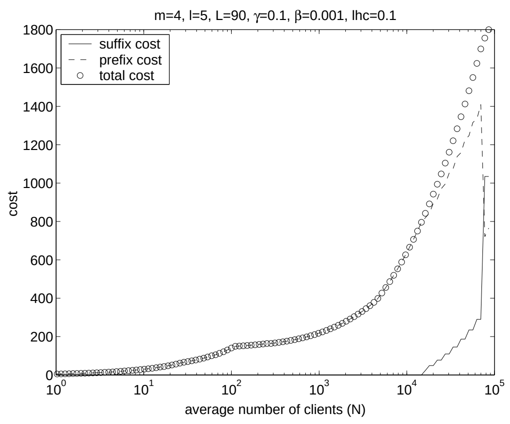


(a) total system cost

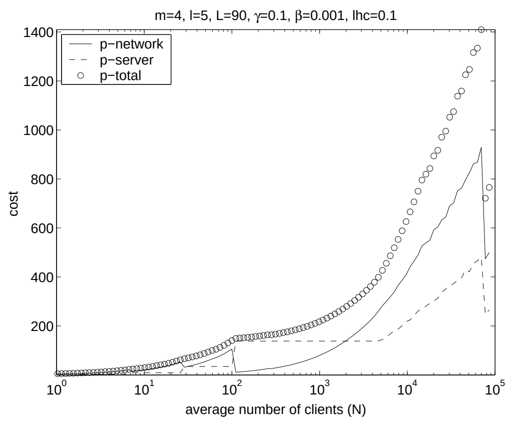


(b) prefi x cost breakdown

Figure 2.12: Breakdown of costs for $\gamma = 0.1$ and $lhc = 1$.



(a) total system cost



(b) prefi x cost breakdown

Figure 2.13: Breakdown of costs for $\gamma = 0.1$ and $lhc = 0.1$.

2.4.5 Ignoring Server Costs

Previous studies of video distribution schemes [22] have often ignored the server cost and only considered the network cost. We can model this case if we choose $\gamma = 0$. When we ignore the server cost, placing the prefix servers at the leaves is always optimal since the prefix *network* cost is minimized in this case.

We plot the optimal prefix length for $\gamma = 0$ in figure 2.14(b). For both values of $lhc = \{1, 0.1\}$, the optimal prefix comprises the entire video for a large interval of the values of N. For large $N > 10^4$, the optimal prefix length becomes shorter as the centralized suffix system becomes more bandwidth efficient (despite the fact that the video must traverse 5 hops for the suffix as compared to 1 hop for the prefix) than the prefix transmission via *Controlled Multicast*. From figure 2.14(b), if we compare the optimal values for the prefix length with the case of uniform link costs (i.e. $lhc = 1$) and large N ($N > 10^4$) , we see that for $\gamma = 0$ the optimal values are only unnoticeably larger than for the case of $\gamma = 0.1$.

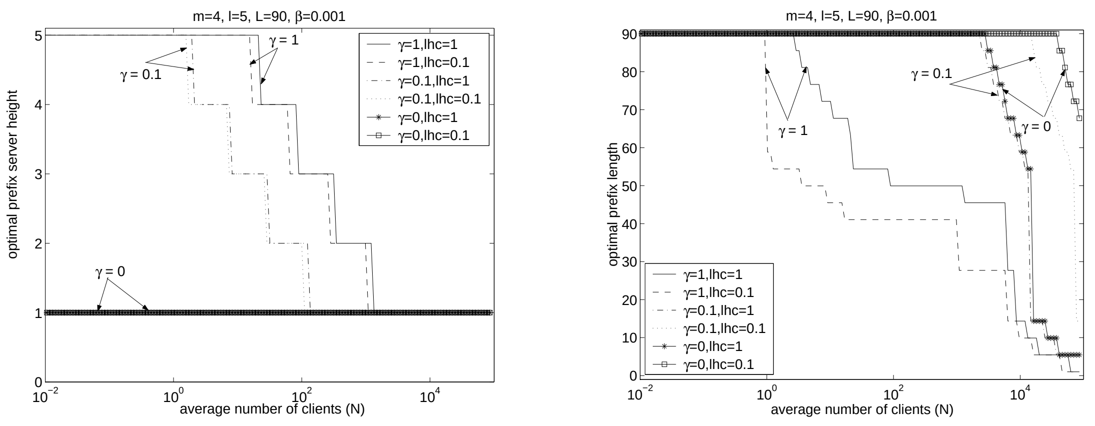
We plot the optimal prefix server height and optimal prefix lengths for all configurations covered so far in figure 2.14. We see how the system adapts the partitioning into prefix and suffix and the placement of the prefix servers as a function of the cost for the network and server resources. When the cost for the servers relative to the cost for network transmission is reduced ($\gamma < 1$), more prefix servers are deployed. For a given video popularity N, this happens in two ways

- The prefix servers are placed closer to the clients (see figure 2.14(a))
- The prefix is made longer (see figure 2.14(b))

In the case of $\gamma = 0$, the entire video is, over a wide range of the video popularity N, delivered by the prefix servers.

We conclude this section by showing in figure 2.15 the optimal total cost values under each scenario discussed. We see that

- for the topology $m = 4, l = 5$ chosen, the majority of the links are at the last-hop to the leaves. Therefore, for non-uniform link costs ($lhc = 0.1$) the cost of the overall system is considerably much smaller than for $lhc = 1$.
- There is surprisingly little difference in the overall cost of the system for $\gamma = 0.1$ and $\gamma = 0$ since in both cases it is the cost for the network transmission that dominates the total cost. For $\gamma = 0.1$,



(a) optimal prefi x server height (b) optimal prefi x length

Figure 2.14: Optimal prefix server height and optimal prefix length for $\gamma = \{1, 0.1, 0\}$ and $lhc = \{1, 0.1\}$.

the impact of the prefix *server* cost is attenuated (for $1 < N < 100$) by using fewer prefix servers (see figure 2.14(a))

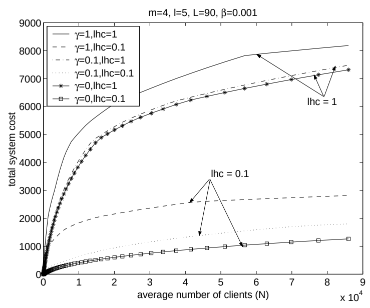


Figure 2.15: Comparison of total system costs for $\gamma = \{1, 0.1, 0\}$ and $lhc = \{1, 0.1\}$.

2.4.6 Conclusions so Far

We have seen how our distribution architecture gives us the cost-optimal configuration as a function of the video popularity N. For a non-popular video, the prefix server is placed at the root[3] and the length of the prefix comprises the whole duration of the video. As the video popularity N increases, the optimal prefix length becomes shorter and the optimal height for prefix servers becomes closer to the clients.

Using the suffix server at the root of the distribution tree becomes economically very interesting for very popular videos, where the prefix length becomes much shorter than the whole duration of the video. However, the optimal suffix length used is quite sensitive to changes in the popularity (see figure 2.14(b) for $10^4 \leq N \leq 10^5$). Therefore, to operate the distribution system in a cost-optimal fashion, one needs to periodically estimate the current popularity of a video and adapt, if necessary, the prefix length.

[3]With the exception of $\gamma = 0$, where the prefi x servers are always placed at height $h = 1$.

2.5 Conclusions and Outlook

2.5.1 Summary

We have presented a scalable video distribution architecture that combines open-loop and closed-loop schemes and assures a zero start-up delay. Under this architecture, each video is split into two parts, the prefix and the suffix. The prefix is transmitted via *Controlled Multicast* (closed-loop scheme) while the suffix is transmitted via *Tailored Periodic Broadcast* (open-loop scheme). Our architecture is very cost-effective since the cost for the prefix transmission increases only with the square root of the number of clients and the suffix distribution cost is, at high request rates, independent of the number of clients and simply a function of the number of segments the suffix is decomposed into.

Another advantage is that our architecture is highly scalable in terms of serving incoming client requests. A client that wants to receive a video contacts its responsible prefix server. The central server only needs to know if there is at least one client connected, which it can learn by communicating with the prefix servers. This interaction between the clients and the servers avoids having a bottleneck due to handling all the requests by a single server.

Furthermore, we have developed an analytical cost model, called PS-model, that allows us to minimize the operational cost of our architecture. In the cost model, we include not only the network bandwidth cost, but also the costs for the server I/O bandwidth and server storage. Using the PS-model we can determine the

- Bandwidth and streaming costs for prefix and suffix transmissions,
- Optimal prefix length, and
- Optimal position of the prefix servers.

PS-model makes the tradeoff between the server and network bandwidth costs to determine the cost-optimal prefix length and the optimal prefix server placement. As key results, we found that

- The cost efficiency increases with increasing video popularity N. For a 10-fold increase in N from 9,000 to 90,000 clients, the total system cost only doubles.
- A popular video is replicated at many prefix servers that are placed close to the clients.
- A non-popular video is placed at the root or very close to the root to reduce the server storage cost incurred by replicating the prefix across many prefix servers.
- The central suffix server is highly efficient to serve very popular videos for which the optimal suffix comprises the major portion of the video.

2.5.2 Outlook

In this chapter we analyzed the efficiency and scalability of our architecture for video delivery. Still, there are many extensions to the PS architecture that are worth to be studied. In the next chapter we extend the PS-model and look at the following scenarios: (i) The increase in the system cost for non optimal placement of the prefix servers (i.e. when the prefix servers can not be placed at any level through the network), (ii) How our architecture operates in the case of short videos of a few min (clips), (iii) The dimensioning of a video on demand service from scratch, (iv) The case of the optimization of an already installed video on demand service (i.e. the limited resources scenario), (v) The use of satellite to broadcast the suffix part of the video, and (vi) Allowing clients to store locally the prefix of some/all popular videos.

Chapter 3

Extensions to the PS Architecture

3.1 Introduction

The PS-model was basically designed to compute the optimal transmission cost for a video with a given popularity. However, the use of the PS-model is not limited only to this purpose. In this chapter, we show that the PS-model can be used to cover many interesting scenarios that address practical aspects while deploying a large scale VoD service:

- **Costs for Non-optimal Prefix Server Placement**
  In the previous chapter, we assumed that prefix servers can be deployed at any level throughout the network. However, this may not always possible in practice. We apply the PS-model to derive the increase in the system cost for the case where the prefix servers can be placed only at fixed levels in the hierarchy (section 3.2).

- **Short Videos**
  The results that we gave so far were all for the case of a long video (i.e. movies). Besides movies, there exist also short videos which correspond to news clips or clips for product promotions. We address whether it is also cost-efficient to divide short videos into a prefix and a suffix (section 3.3).

- **Provisioning**
  So far we have looked at the case of a single video. We studied how to determine the optimal prefix length and the optimal prefix placement as a function of the video popularity N. However, a complete video distribution system will offer to clients a host of different videos $\mathcal{V} = \{1, \ldots, K\}$ to choose from. The PS-model can be used to solve the provisioning problem for a given set of videos whose popularities are known: We just need to execute the model for each video separately to determine the optimal prefix length and the placement of the prefix servers (section 3.4.2).

- **Video Assignment Problem for an Existing Configuration**
  Very often, the situation will be such that the video distribution system has been already deployed, i.e. the central suffix server and the prefix servers have been installed and changing the location (placement) or the capacities of the prefix servers is not possible. Given that the placement and the capabilities of the prefix servers are *fixed*, we extend our cost model to determine the cost-optimal prefix length and prefix placement for a set of videos and popularities (section 3.4.3).

- **Evaluation of architectural choices**
  Today, digital VCRs with several tens of Gigabyte of local storage are commercially available [6]. Given a local storage capacity, one can proactively download the prefixes of the most popular

videos directly into the VCR. We extend the PS-model to evaluate the overall cost reduction due to the use of the local storage in the VCR (section 3.5.2).

Another attractive architectural option would be the use of satellite to distribute the suffix part of the video. Nowadays, satellites have become a very cost effective transmission medium for sending data to a group of users. We adapt the PS-model and derive the scenarios under which such an option is of benefit to the system (section 3.5.1).

3.1.1 Organization of this Chapter

In what follows, we discuss the non-optimal placement scenario for the prefix servers in section 3.2. Section 3.3 proves that the PS-model is still advantageous in the case of short videos, e.g. news clips. In section 3.4, we first study the dimensioning of the system with K videos and no resource constraints. We next devise the algorithm that optimally assigns the K video to prefix servers with limited I/O and storage resources (section 3.4.3). In section 3.5 we extend the PS-model to evaluate the benefit of two architectural choices (i) The use of satellite to broadcast the suffix part of the video and (ii) Providing clients with set-top boxes to store locally the prefix part of some/all popular videos. We conclude this chapter in section 3.6.

3.2 Costs for Non-optimal Prefi x Server Placement

3.2.1 Introduction

For a large video distribution system that services many movies, it will be feasible to place prefix servers at every level of the distribution tree; however, this may be impossible for smaller systems. Thereby, it is worth evaluating the cost performance of a video distribution system where the prefix servers are placed non-optimally. We will examine how the system adjusts the prefix length to find the most cost-efficient solution for the case where

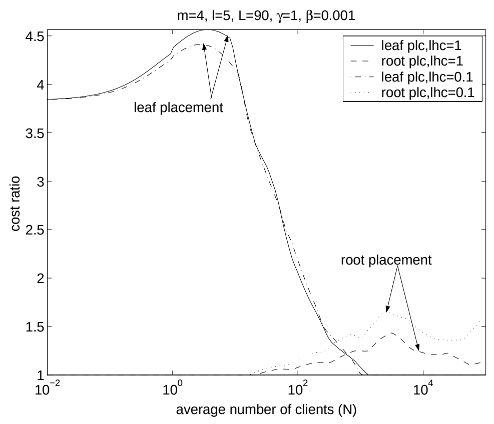
- The prefix server is placed at the root, which will be referred to as **root placement**
- The prefix servers are placed at the leaves (i.e. at height h = 1, one level above the clients), which will be referred to as **leaf placement**

We always assume that the cost for the server is taken into account (i.e. $\gamma \neq 0$) and consider the following scenarios

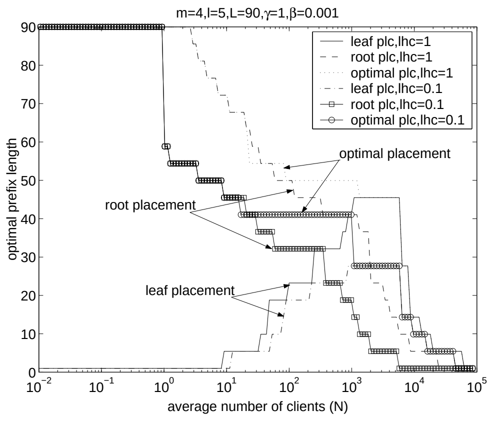
- Network is cheap, $\gamma = 1$ and $lhc = \{1, 0.1\}$.
- Network is expensive, $\gamma = 0.1$ and $lhc = \{1, 0.1\}$.

3.2.2 Network is Cheap

We first discuss the case with $lhc = 1$, where the per-link network costs are the same across the network. We know from figure 2.7(a) that for values of $N < 20$, the optimal placement of the prefix server is at the root. For increasing values of $N > 20$, the optimal placement finds the best tradeoff between network and server cost by moving the prefix servers closer to clients and reducing the length of prefix. When we fix the placement of the prefix server at the root, the only degree of freedom left to minimize the cost is to adjust the prefix length. We see in figure 3.1(b) that the prefix length for the root placement case decreases more rapidly than when the prefix placement is chosen optimally. In figure 3.1(a) we plot the ratio of the total system costs. For the case when the prefix server is always at the root as compared



(a) Cost ratio for prefix server, $\gamma = 1$, $lhc = \{1, 0.1\}$



(b) Optimal prefix length, $\gamma = 1$, $lhc = \{1, 0.1\}$

Figure 3.1: System cost ratio for non-optimal placement to optimal placement system cost and optimal prefix length for $\gamma = 1$ and $lhc = \{1, 0.1\}$.

to when it is optimally placed, the total system cost increases by up to 60%: For very popular videos, placing the prefix server at the root results in a longer suffix in order to limit the high cost for the prefix distribution.

When the prefix servers are always at the leaves, the cost increase as compared to the optimal placement is much higher than for the root placement and can be up to 450%. The reason is that the leaf placement is very expensive for non-popular videos ($N < 10^2$). This big difference in cost is due to the fact that keeping copies of a video in all the prefix servers placed at the leaves is very inefficient for videos that are not very popular, i.e. $N < 10^3$. When the prefix servers are placed at the leaves and the video popularity is low ($N < 10$), the system chooses the minimal possible prefix length of $D = 1$ min[1] to limit the overall cost of keeping many copies of that prefix (see figure 3.1(b)). For larger values of N, the optimal prefix server placement is at the leaves, and so the performance of the non-optimal system is the same as the optimal one for $N > 10^3$.

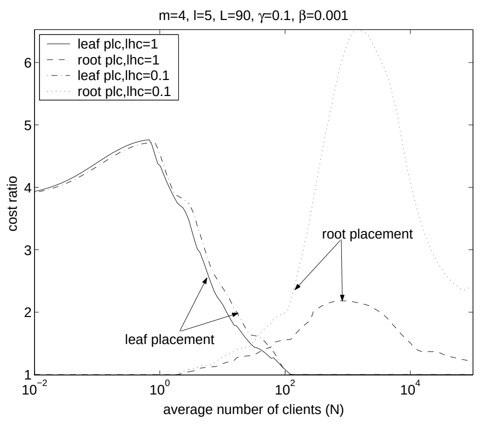
When we compare the case where all links of the network have the same cost ($lhc = 1$) to the case where the last-hop links are less expensive for $lhc = 0.1$, we see a little difference in the results. For $lhc = 0.1$, the worst case cost increase due to leaf placement is a little bit less than for $lhc = 1$. For the root placement, the fact that the last-hop links are less expensive ($lhc = 0.1$) increases the cost of root placement as compared to $lhc = 1$.

Overall, when the network is cheap and we must choose between root and leaf placement, root placement is preferable.

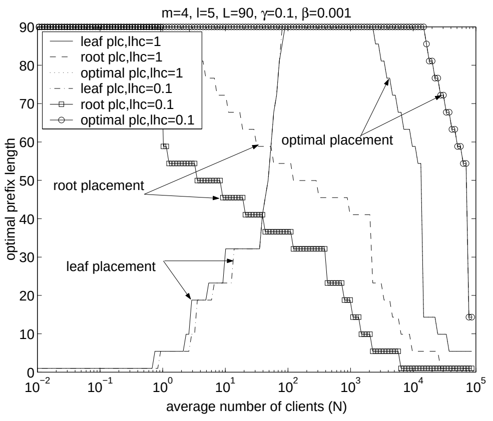
3.2.3 Network is Expensive

We examine now the opposite situation, where the network is expensive as compared to the cost for servers (see figure 3.2(a)). When the network is expensive, the worst case cost performance of leaf placement deteriorates only slightly as compared to the case when the network is cheap, since the cost for the servers placed at the leaves dominates the total system cost. For root placement, the worst case cost is significantly higher as in the case of $\gamma = 0.1$, in particular for $lhc = 0.1$, since the network transmission has become more expensive, which is directly reflected in an increase of the total system cost. The optimal placement for $\gamma = 0.1$ moves the prefix servers for increasing N rapidly towards the

[1]We limit the minimal length of the prefix to 1 min in order to provide a zero start-up delay service



(a) cost ratio for prefix server, $\gamma = 0.1$, $lhc = \{1, 0.1\}$



(b) Optimal prefix length, $\gamma = 0.1$, $lhc = \{1, 0.1\}$

Figure 3.2: System cost ratio for non-optimal placement to optimal placement and optimal prefix length for $\gamma = 0.1$ and $lhc = \{1, 0.1\}$.

clients and chooses a prefix that comprises the entire video for all except the very popular ones $N > 10^4$ (see figure 2.14). Since the root placement can not move the prefix servers, it shortens the prefix length drastically with increasing N to offset the cost-increase due to the prefix placement at the root (see figure 3.2(b)).

3.2.4 Conclusion

The video delivery architecture has normally two degrees of freedom: the prefix length and the prefix server placement, which can be varied to determine the cost optimal solution. When we remove one degree of freedom and fix the placement of the prefix servers, the total system cost can increase significantly, in particular for the case of leaf placement, where the server cost dominates as compared to the network cost.

3.3 Short videos

3.3.1 Introduction

So far we have looked at videos of length $L = 90$ min, which corresponds to feature movies. Besides movies, there are news clips or clips for product promotions, which are much shorter in length and can also be delivered via a video distribution system. Thus, it would be interesting to evaluate how efficiently the video distribution architecture supports the distribution of these clips. We will consider clips of $L = 7$ min[2]. As before, the network has an outdegree m = 4 and a number of levels l = 5. We vary the popularity of the video between $10^{-2} \leq N \leq 10^5$.

The optimal configuration is computed using the PS-model. The PS-model allows to determine which fraction of the video should be stored at the prefix servers and where to place the prefix servers to minimize the delivery cost. In addition, we consider two more cases where we remove one degree of freedom:

[2]We also looked at clips of 2 min of length. The results that we obtained are very similar to the ones for $L = 7$ and are therefore not present here.

- $D = L$, i.e. the prefix comprises the full video, which we refer to as **full video caching**. However, the height of the prefix servers is chosen such to minimize the overall system cost.

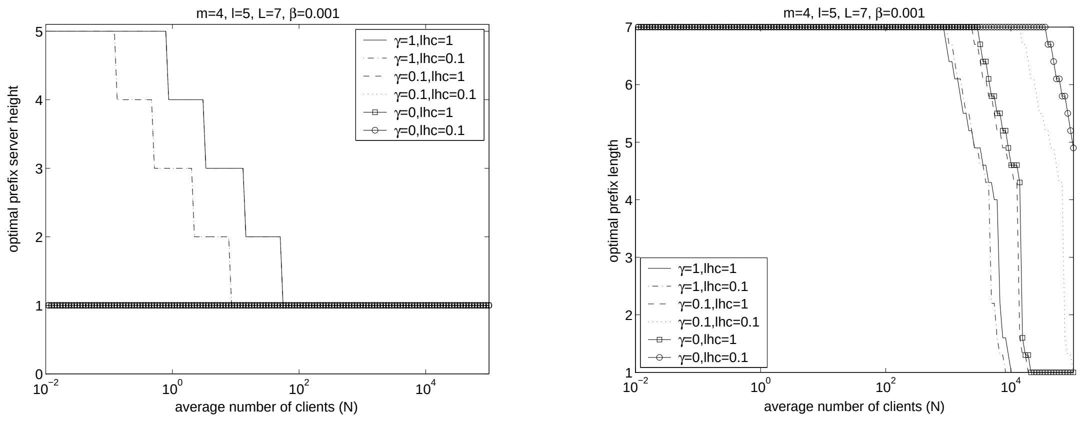
- The prefix server is fixed at the root, which we introduced in section 3.2 as **root placement**. However, the length of the prefix is chosen such to minimize the overall system cost.

3.3.2 Results

Figure 3.3 shows the optimal prefix server placement and the optimal prefix length in the hierarchy as a function of the video popularity N. A comparison with the values obtained for long videos of $L = 90$ min (see figure 2.14) shows that the video distribution system behaves the same way, for both short and long videos:

- With increasing video popularity N, the placement of the prefix servers moves closer to the clients

- For all but the most popular videos, the optimal prefix comprises the whole video.

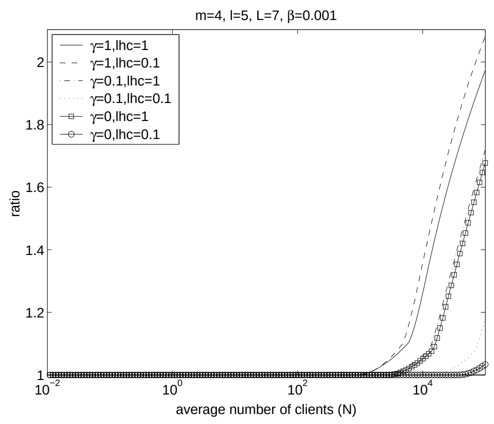


(a) optimal prefi x server height (b) optimal prefi x length

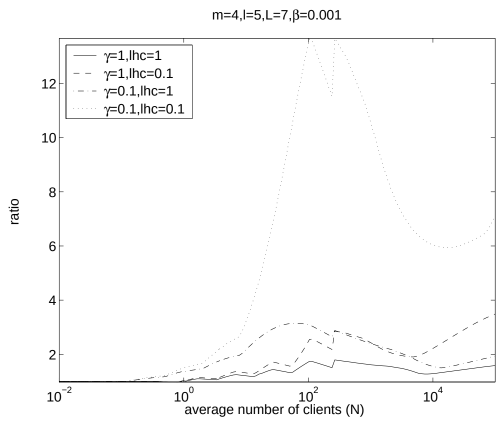
Figure 3.3: Optimal prefix server height and prefix length for $\gamma = \{1, 0.1, 0\}$, $lhc = \{1, 0.1\}$, and $L = 7$ min.

As we can observe from figures 3.3(b) and 3.4(a), full video caching is the optimal solution, except for the most popular videos. In Figure 3.4(b) we plot the ratio of the delivery cost of a video obtained in the case of root placement as compared to the delivery cost obtained when both, the prefix length and the prefix server placement are chosen optimally[3]. We see that there is an additional cost of fixing the prefix server at the root for all values of the video popularity N except for very small ones, where placing the prefix server at the root is the optimal choice. The additional cost due to the root placement is (i) *lowest* when the network transmission is cheap ($\gamma = 1$) and (ii) *highest* when the relative cost for the prefix servers is low ($\gamma = 0.1$) **and** the transmission cost over the last hop is discounted ($lhc = 0.1$). When the network is expensive ($\gamma = 0.1$), the cost ratio is worst for the values of N where the optimal prefix server placement puts the prefix servers at the leaves ($h = 1$) and chooses a prefix that comprises the entire video ($D = L$). The shape of the curves is similar to the ones observed for long videos (see figures 3.1(a) and 3.2(a)).

[3]We do not plot the cost ratio for $\gamma = 0$, which can reach a value up to 40 for small values of N.



(a) Cost ratio for full video caching



(b) Cost ratio for root placement

Figure 3.4: Cost ratio for full video caching system and cost ratio of root placement for $\gamma = \{1, 0.1, 0\}$, $lhc = \{1, 0.1\}$, and $L = 7$ min.

3.4 Video Distribution System for a Set of Videos

3.4.1 Introduction

For a set $\mathcal{V} = \{1, \ldots, K\}$ of videos, the PS model computes separately the optimal system cost of each video $i \in \mathcal{V}$. The total system cost is the sum over the system costs of all K videos in $\mathcal{V}$. In the following, we assume that the popularity of the videos follows a Zipf distribution [86], $N_i = \frac{A}{i^\alpha}$, $i \in \mathcal{V}$, where the normalization constant A determines the popularity of the most popular video and α the slope of the popularity distribution; the bigger α, the steeper the slope, i.e. the more biased or skewed the popularity distribution (figure 3.5).

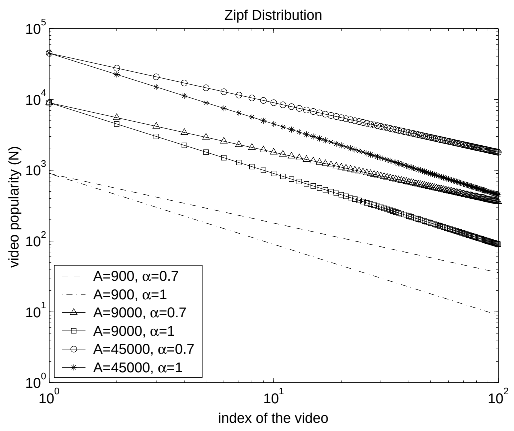


Figure 3.5: The popularity N_i as a function of the index i according to a Zipf distribution.

3.4.2 Provisioning of the Prefix Servers

The aim of provisioning is to compute the resources required in terms of video server storage and video server I/O bandwidth for a given set of videos $\mathcal{V} = \{1, \ldots, K\}$ with popularities N_i ($\lambda_i = \frac{N_i}{L}$), for

$i \in \mathcal{V}$[4]. We will concentrate here on the provisioning of the prefix servers inside the network. However, the provisioning of the servers at the root can be done in a similar way.

We use equation 2.2 (section 2.3.4) to compute the optimal prefix length D_i, the optimal threshold T_i, and the optimal prefix server level l_i for each video $i \in \mathcal{V}$. Let $\mathcal{L}(j) \triangleq \{i \in \mathcal{V} \mid l_i = j\}$ denote the subset of videos whose prefix will be optimally served by the prefix servers of level j. I/O_i denotes the amount of I/O bandwidth needed for each prefix server at level j to serve the prefix of video $i \in \mathcal{V}$. The value of I/O_i can be computed the same way as $C_{I/O}^{prefix}$ in section 2.3.2:

$$I/O_i = b \cdot \frac{D_i + \frac{T_i^2 \lambda_i}{2 \cdot m^{l-i}}}{T_i + m^{l-i}/\lambda_i}$$

At each level j, the total storage $[PS_{st}(j)]$ and I/O $[PS_{io}(j)]$ capacity of the prefix servers are computed as the sum of the prefix lengths and the amount of I/O bandwidth needed over all prefixes placed at that level. Hence, at level j, the resource requirements of the prefix servers are given by:

$$\begin{aligned} PS_{st}(j) &= \sum_{i \in \mathcal{L}(j)} b \cdot 60 \cdot D_i \qquad \forall j \in \{1, \ldots, l-1\} \\ PS_{io}(j) &= \sum_{i \in \mathcal{L}(j)} I/O_i \quad \forall j \in \{1, \ldots, l-1\} \end{aligned}$$

Since we assume a homogeneous client population, the load will be uniformly distributed over the prefix servers at a particular level. Therefore, all prefix servers at a particular level j will have the same capabilities.

3.4.3 Assignment of a Set of Videos to Prefix Servers with Limited Storage and I/O Capabilities

We now consider the case where the prefix servers have been installed and that it is not possible to add new prefix servers or to modify their placement in the distribution hierarchy. The values of $PS_{st}(j)$ and $PS_{io}(j)$ are known $\forall j \in \{1, \ldots, l-1\}$. We will refer to them as **prefix server constraints**. However, we allow for modifications to the servers installed at the *root*. This means that there are no constraints on the resources of the central suffix server or the prefix server at the root ($l = 0$).

To solve the prefix assignment problem for a set of videos $\mathcal{V}$, we need to find the placement that satisfies the prefix server constraints of the system and minimizes the total system cost. We formulate the assignment problem for a set $\mathcal{V}$ of videos as follows:

$$\begin{cases} \min\limits_{\theta_{ij}} (\sum\limits_{j=0}^{l-1} \sum\limits_{i \in \mathcal{V}} C_{PS}^{system}(i,j) \times \theta_{ij}) & \\ s.t. \quad \sum\limits_{i \in \mathcal{V}} D_{ij} \times \theta_{ij} \leq PS_{st}(j) & 1 \leq j \leq l-1 \\ \qquad \sum\limits_{i \in \mathcal{V}} I/O_{ij} \times \theta_{ij} \leq PS_{io}(j) & 1 \leq j \leq l-1 \\ \qquad \sum\limits_{j=0}^{l-1} \theta_{ij} = 1, & \forall\, i \\ \qquad \theta_{ij} \in \{0,1\}, & \forall\, i,j \end{cases}$$

[4]For sake of simplicity we assume that all videos have the same length L.

where (i) $C_{PS}^{system}(i,j)$ is the lowest total system cost achievable when placing the prefix of video i at level j, (ii) D_{ij} is the corresponding optimal prefix length, (iii) I/O_{ij} is the corresponding amount of I/O bandwidth needed to schedule the prefix of video i from level j, and (iv) θ_{ij} is a binary variable. θ_{ij} is equal to 1 if the prefix of video i is placed at level j and 0 otherwise. $\sum_{j=0}^{l-1} \theta_{ij} = 1$ indicates that no video prefix can be stored at more than one level in the hierarchy. Both, the objective function and the constraints are linear functions of the binary variables θ_{ij}. We resolve this optimization problem using *XPress-MP* [29], a dynamic programming package.

Results Moving a video prefix from an optimal level to a non-optimal one in order to satisfy the constraints increases the delivery cost of the video and consequently the overall system cost. It is interesting to evaluate how the prefix server constraints will impact the system cost of the video distribution system. To this purpose, we compute the system cost C_{opt}^{vds} *without* any constraints and the system cost C_{constr}^{vds} with prefix server constraints, which are defined as

$$\begin{aligned} C_{opt}^{vds} &= \sum_{i \in \mathcal{V}} C_{PS}^{system}(i) \\ C_{constr}^{vds} &= \sum_{j=0}^{l-1} \sum_{i \in \mathcal{V}} C_{PS}^{system}(i,j) \times \theta_{ij} \end{aligned}$$

We use the cost ratio $C_{constr}^{vds}/C_{opt}^{vds}$ to evaluate how well the video distribution architecture can adapt to changes in the video popularity and the number of videos. We plot in figure 3.6 the cost ratio for $\alpha = \{0.2, 0.6, 1\}$ and different numbers of videos $K = \{100, 120, 150\}$. We set the normalization constant to $A = 9000$. The prefix server constraints were computed for the case of $K = 100$ videos with $\alpha = 0.6$ for the Zipf distribution. This initial distribution ($\alpha = 0.6$) is skewed or biased enough to meet small systems. Note that for the rest of this chapter, the length of the video will be always $L = 90$ min. Figure 3.6 shows the following interesting features:

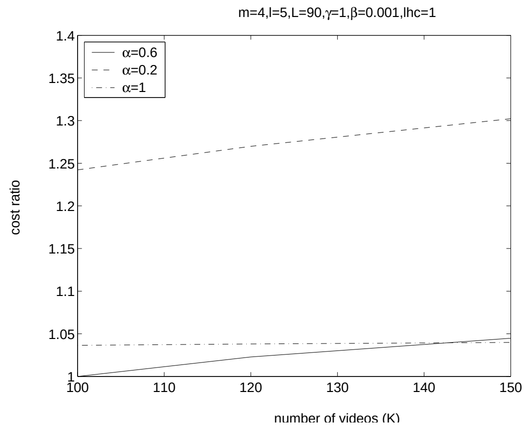


Figure 3.6: Cost ratio $C_{constr}^{vds}/C_{opt}^{vds}$.

- The cost ratio increases sub-linearly with the number of videos.
- The increase in the system cost strongly depends on the parameter α.

These results fit well with the intuition: (i) The larger the popularity N of a video, the closer is the placement of the prefix to the clients and (ii) If there is no more place for a popular video at the higher levels of the tree hierarchy (i.e. close to clients), the video must be moved closer to the root.

For a given Zipf distribution, increasing the number of videos adds videos at the *tail* of the distribution. Those additional videos are the least popular ones, and must be moved near the root to free resources for the more popular ones at the higher levels. The placement of the least popular videos close to the root will use up very few of the constraint resources, which explains the small change in the cost ratio with increasing number of videos. At some point, all the additional videos will have such a low popularity that their prefixes will all be placed at the root and the cost ratio will no longer change with K.

For $\alpha = 1$, the popularity distribution is very skewed, and increasing the number of videos K in the system has no impact on the cost ratio since those additional videos are all optimally placed at the root. If we compare $\alpha = 0.2$ and $\alpha = 0.6$, we find that, as the number of videos K increases, the cost ratio increases more rapidly for $\alpha = 0.2$ than for $\alpha = 0.6$. The reason is that, for $\alpha = 0.2$, the videos added to the system are relatively more popular than those for $\alpha = 0.6$, and as a consequence, moving them closer to (or placing them at) the root to satisfy the constraints comes out more costly.

We also evaluated other scenarios such that, $\gamma = 0.1$ or $\gamma = 0$ and for a discounted cost of the last hop link $lhc = 0.1$. The results that we obtained were similar to what we have showed and are therefore are not given here.

Conclusion The PS-model adapts itself very well in case of limited prefix server resources. Non-popular videos are moved close to the root to free a place for the more popular ones. For $\alpha = 0.6$ and $\alpha = 1$, the increase in the overall system cost for increasing number of videos is very low, less than 5%. Even when the popularity distribution differs a lot from the one used when determining the prefix server resources (as is the case for $\alpha = 0.2$), the extra cost incurred will be small, 25–30%.

3.5 Evaluation of architectural choices

3.5.1 Use of Satellite for Suffi x Transmission

Introduction

Satellites are a very cost effective transmission medium for sending data to a group of users. The cost of 1 Mb/month satellite transmission bandwidth is about \$ 10,000 [63] whereas the cost for 1 Mb/month terrestrial transmission bandwidth is \$1300 for 1 Mbps during one month in case of a T1 line and \$350 for 1 Mbps during one month in case of a OC-48 transmission link [15]. We now consider the case where the *suffix* is transmitted via satellite directly to the clients, while the prefix, as before, is transmitted from the prefix servers to the clients via a terrestrial network. We refer to this model as the **S-sat** model. Compared to the PS-model, all cost terms (see tables 2.1, 2.2) stay the same except C_{net}^{suffix} and $C_{I/O}^{suffix}$. For the S-sat model, the suffix server transmits the suffix periodically and continuously regardless of the number of clients. As a consequence, the cost terms for C_{net}^{suffix} and $C_{I/O}^{suffix}$ for S-sat model are:

$$\begin{aligned} C_{net}^{suffix} &= \sigma * R_t^{min} \\ C_{I/O}^{suffix} &= R_t^{min} \end{aligned}$$

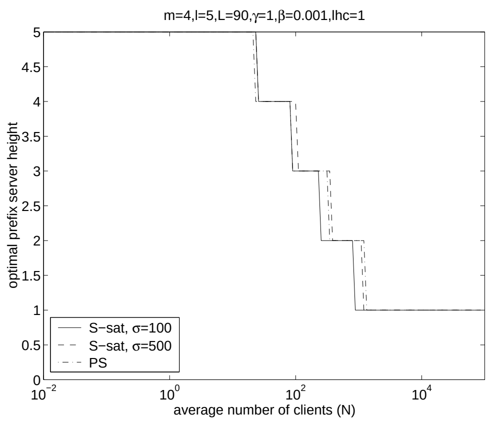
where σ is a weight factor that allows to express the cost for the satellite transmission in *relative* terms with respect to the other cost elements such as terrestrial transmission or server storage and I/O. In the following, we will use two different values for σ namely $\sigma = 100$ and $\sigma = 500$. For $\sigma = 500$, the cost of the satellite transmission is 5 times as high as for $\sigma = 100$. In the light of the absolute prices given above, we consider both values as "conservative" in the sense that they are likely to overestimate the cost of satellite transmission compared to terrestrial transmission: From equation 2.1 we know that the costs for the PS-model are $C_{netw}^{suffix} = R_t^{min} \sum_{j=1}^{l} m^j P(j)$ and $C_{I/O}^{suffix} = R_t^{min} \cdot P(0)$. If the number of clients is large enough such that all links of the suffix distribution tree are occupied all the time, we have $P(j) = 1, \forall j$, and these expressions simplify to

$C_{netw}^{suffix} = R_t^{min} \sum_{j=1}^{5} m^j$ and $C_{I/O}^{suffix} = R_t^{min}$. For $m = 4, l = 5$, we get $\sum_{j=1}^{5} 4^j = 1364$.

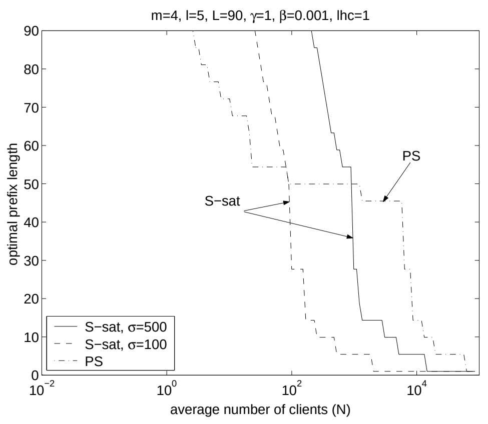
Results

We see in figure 3.7(b) that for the S-sat model, the prefix decreases more rapidly with increasing N since the transmission of the suffix via satellite is more economic compared to a transmission over the terrestrial network. The smaller the value of σ, the cheaper the satellite transmission and the more cost effective the S-sat model. For values of $N > 10^2$, the S-sat model can be very cost effective (see figure 3.8(a)). For very high numbers of clients, the suffix becomes eventually as large as possible (89 min) and satellite suffix transmission can reduce the cost by up to 80% (for $\gamma = 1, lhc = 1, \sigma = 100$). The cost reduction obviously depends on the relative cost of a satellite transmission as compared to a terrestrial transmission. For the case $\sigma = 500$ and $\gamma = 1, lhc = 0.1$, the satellite transmission is quite expensive compared to a terrestrial transmission and as a result, the suffix satellite transmission is not competitive.

When we look at the cost breakdown in figure 3.8(b), we see that for $N > 10^3$ the suffix cost is no longer increasing with increasing N since the suffix has its maximal possible length (89 min). On the other hand, the prefix cost keeps increasing and eventually dominates the overall distribution cost.



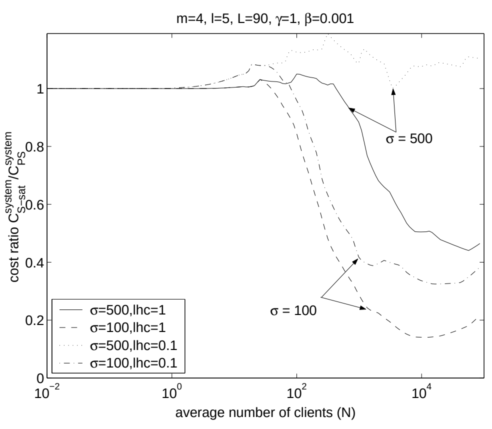
(a) optimal prefi x server height



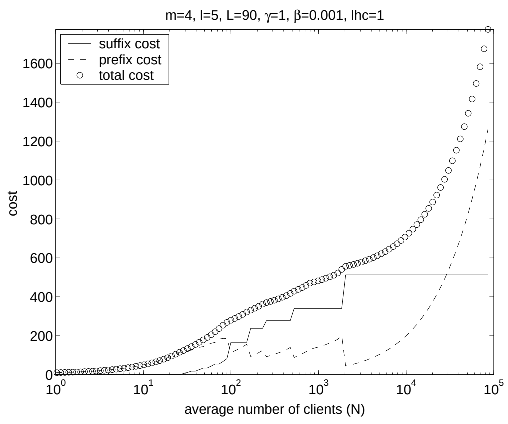
(b) optimal prefi x length

Figure 3.7: Optimal prefix server height and optimal prefix length for both, PS and S-sat models, as a function of the number of clients N for $\gamma = 1$, $lhc = 1$ and $\sigma = \{500, 100\}$.

On the other hand, as we have seen in figure 2.14 for the PS model, $\gamma = 0.1$ makes the prefix servers cheaper, which allows to use more of them. Such a cost reduction of the prefix servers also benefits the S-sat model. As a consequence, for $\gamma = 0.1$, the satellite transmission still remains, for large values of N, much more cost effective than the transmission of the suffix via terrestrial links. If we completely ignore the server cost ($\gamma = 0$) and the last hop cost is reduced ($lhc = 0.1$), suffix transmission via satellite will remain more cost effective provided that satellite transmission is cheap ($\sigma = 100$).

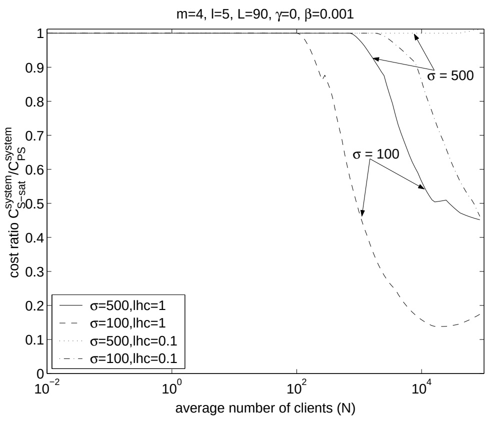


(a) $C^{system}_{S-sat}/C^{system}_{PS}$, $\gamma = 1$ and $lhc = \{1, 0.1\}$

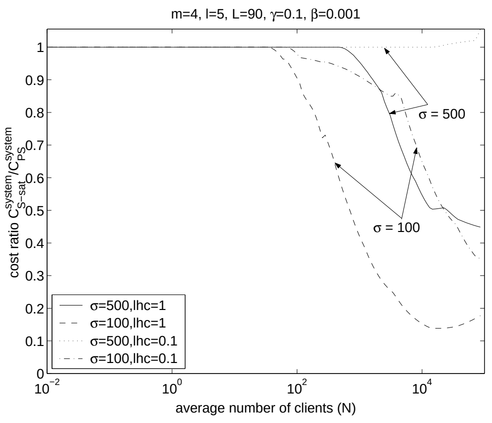


(b) System cost breakdown for $S - sat$, $\sigma = 100$, $\gamma = 1$ and $lhc = 1$

Figure 3.8: Cost ratio ($C^{system}_{S-sat}/C^{system}_{PS}$) for $\gamma = 1$, $lhc = \{1, 0.1\}$ and $\sigma = \{100, 500\}$ and breakdown cost for S-sat for $\sigma = 100$.



(a) $lhc = 1$



(b) $lhc = 0.1$

Figure 3.9: Cost ratio ($C^{system}_{S-sat}/C^{system}_{PS}$) for $\gamma = \{0.1, 0\}$, $lhc = \{1, 0.1\}$ and $\sigma = \{100, 500\}$.

Conclusion

Using the satellite for suffix transmission with prefix servers placed close to the clients at height $h = 1$ allows to distribute highly popular videos in a very cost effective way. For the different scenarios considered

- Using satellite suffix distribution as compared to terrestrial links has very little (for $\gamma = 1$, see figure 3.7(a)) or no (for $\gamma = 0.1$ and $\gamma = 0$, figure not shown) impact on the placement of the prefix servers.

- The cost comparison between S-sat and PS is very insensitive to the value of γ, which determines the relative cost between server and terrestrial transmission.

- When satellite transmission is cheap ($\sigma = 100$), the cost-optimal solution is to choose a large suffix length. In this case the S-sat model can reduce the distribution cost of popular videos by more than 50% .

- When the relative cost of satellite transmission is high, i.e. for $lhc = 0.1$ and $\sigma = 500$, the PS-model is at least as cost effective as the S-sat model.

3.5.2 Set-top Box at the Client Side for Prefix Storage

Introduction

Today, set-top boxes at the clients side provide a large amount of storage capacity at a low cost. For instance, the digital video recorder developed by Tivo [6] allows to store between 20 and 60 hours of MPEG II coded video and can receive transmissions at high data-rates. We evaluate the overall cost reduction for the PS model when we allow the client to store locally the prefixes of some videos. We call this new architecture the hybrid architecture (P-hybrid) that in contrast to the PS architecture, allows the prefix to be stored at either the prefix servers or the set-top boxes:

- When the prefix is stored at the prefix servers, the prefix is delivered to clients via *Controlled Multicast* as with the PS model. In this case, the costs for the prefix is the same for both models, P-hybrid and PS ($C^{prefix}_{P-hybrid} = C^{prefix}_{PS}$).
- The prefix can also be downloaded directly to the set-top box. In this case, the prefix cost with the P-hybrid model, as compared to the PS model, is reduced to only the download cost of the prefix.

On the other hand, the suffix is stored at the central server and delivered to clients via *Tailored Periodic Broadcast* as in the case of the PS-model. Thus, the costs for the suffix is the same with both the P-hybrid and the PS models.

Analytical Model

We partition the set of K videos into two subsets, namely S_1 and S_2, with $S_1 \bigcap S_2 = \oslash$. S_1 represents the set of videos whose prefix is stored in the prefix servers. S_2 represents the set of videos whose prefix is stored in the *set-top box*. We calculate separately the cost for the videos in S_1 and S_2. The total P-hybrid system cost is the sum of the system costs over all K videos in the system:

$$C^{system}_{P-hybrid} = \sum_{i=1}^{K} C^{system}_{P-hybrid}(i) = C^{1}_{P-hybrid}(S_1) + C^{2}_{P-hybrid}(S_2) \ .$$

For each video i in S_1, the system cost is computed separately using PS-model, and the cost of S_1 is:

$$C^{1}_{P-hybrid}(S_1) = \sum_{i \in S_1} C^{system}_{PS}(i) \ .$$

Concerning S_2, the suffix cost for each video $i \in S_2$ is computed as in the case of the PS-model since in both architectures the suffix is delivered via *Tailored Periodic Broadcast*. And the prefix cost comprises only the download cost of the prefix:

$$C^{2}_{P-hybrid}(S_2) = \sum_{i \in S_2} C^{prefix}_{P-hybrid}(i) + \sum_{i \in S_2} C^{suffix}_{PS}(i) \ ,$$

with

$$\begin{aligned} C^{suffix}_{PS}(i) &= C^{suffix}_{netw}(i) + \gamma(C^{suffix}_{I/O}(i), \beta C^{suffix}_{sto}(i)) \\ C^{prefix}_{P-hybrid}(i) &= b\,\frac{D_i \cdot 60}{T_d \cdot 60} \sum_{j=1}^{l} m^j = b\,\frac{D_i}{T_d}(\frac{m^{l+1} - m}{m-1}) \end{aligned}$$

where $C^{suffix}_{netw}(i)$, $C^{suffix}_{I/O}(i)$ and $C^{suffix}_{sto}(i)$ are defined as in table 2.2 and T_d (min) is the **download interval** (time between two downloads) of the prefix at the set-top box. To minimize the cost of S_2, we should find the optimal set of prefix lengths $\{D_i\}_{i \in S_2}$ of the videos in S_2. For this purpose, we solve the following problem:

$$\begin{aligned} &\min_{i \in S_2} [C^2_{P-hybrid}(S_2)] \\ &s.t. \quad \sum_{i \in S_2} D_i \leq Cap \end{aligned}$$

Where Cap (min) is the storage capacity of the set-top box. The problem expressed above is a non linear programming problem subject to linear inequality constraints. To obtain a solution, we apply the `fmincon` package of Matlab.

For a given partition of the videos between the two disjoint subsets S_1 and S_2, we apply the PS model to compute the cost C^1_{PS} of S_1 and we apply the fmincon package of Matlab to compute the cost $C^2_{P-hybrid}$ of S_2. The P-hybrid system cost is the sum of costs of S_1 and S_2. However, to find the optimal total system cost, we must find which video prefixes must be stored in the set-top box. We present the following algorithm to find the optimal split of the set of K videos between S_1 and S_2. We sort the videos in decreasing order of popularity ($N_i > N_j$ if $i < j$). We start with the case where all videos are in S_1 (all the prefixes are stored in the prefix servers). In this case, the system cost is equal to the cost of S_1, which is computed with the PS model. Since in S_1 the most popular video consumes the maximum of the system resources available among all videos (see figure 2.6), we move the videos from S_1 to S_2, one after the other, starting with the most popular one in the system. At each step, we compute the following parameters: (i) The new costs of S_1 ($C^1_{PS}(S_1)$) and S_2 ($C^2_{P-hybrid}(S_2)$), (ii) The new cost of the P-hybrid model ($C^{system-new}_{P-hybrid}$), which is the sum of the new costs of S_1 and S_2. For each video i in the system we can determine: (i) The subset to join (S_1 or S_2), (ii) The optimal prefix length D_i, (iii) The optimal level of placement of the prefix in the network. We compare the P-hybrid system costs at the current step $C^{system-new}_{P-hybrid}$ and at the previous step $C^{system-opt}_{P-hybrid}$; if $C^{system-new}_{P-hybrid} \geq C^{system-opt}_{P-hybrid}$ then we stop and the parameter values computed at the previous step are considered to be the optimal ones. The following figure summarizes all the aforementioned steps:

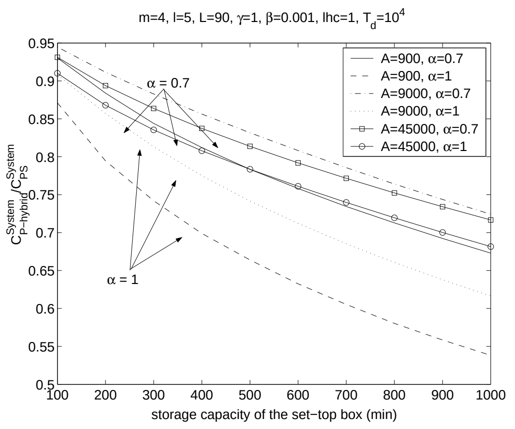
```
For a set $\mathcal{V}$ of $K$ videos
/$\star$ We sort the videos in decreasing order of popularity $\star$/
$\forall\, i, j \in \{1, ..., K\}$,     If $i < j$, then $N_i > N_j$
/$\star$ We start with the case where the prefixes of all $K$ videos are stored in
the prefix servers $\star$/
$S_1 := \{1, ..., K\}$,     $S_2 := \oslash$
$C_{P-hybrid}^{system-opt} := \sum_{i \in S_1} C_{PS}^{system}(i)$
For i := 1:K
    /$\star$ move video $i$ from $S_1$ to $S_2$ $\star$/
    $S_1 := S_1 \backslash \{i\}$,     $S_2 := S_2 \cup \{i\}$
    compute   $C_{P-hybrid}^{system-new} := C_{PS}^{1}(S_1) + C_{P-hybrid}^{2}(S_2)$
    If ($C_{P-hybrid}^{system-new} < C_{P-hybrid}^{system-opt}$) then
        $C_{P-hybrid}^{system-opt} := C_{P-hybrid}^{system-new}$
        If i == K
            $C_{P-hybrid}^{system} := C_{P-hybrid}^{system-new}$
            /$\star$ We have found the optimal partitioning, stop $\star$/
            $S_1 := \oslash$,     $S_2 := \{1, ..., K\}$
            Exit
        endIf
    else
        $C_{P-hybrid}^{system} := C_{P-hybrid}^{system-opt}$
        /$\star$ move back the video $i$ from $S_2$ to $S_1$ $\star$/
        $S_1 := S_1 \cup \{i\}$,     $S_2 := S_2 \backslash \{i\}$
        /$\star$ We have found the optimal partitioning, stop $\star$/
        Exit
    endIf
endFor
```

Results

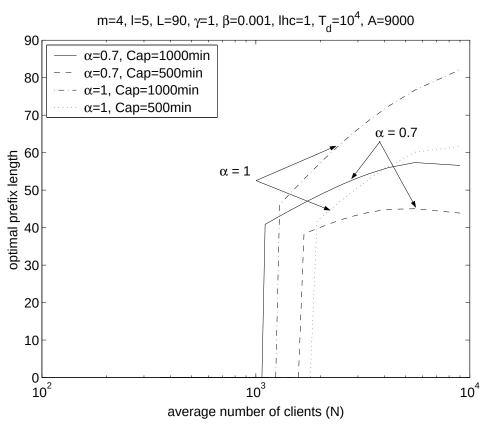
To get new insights into the advantages of having a set-top box at the client side, we plot in figure 3.10(a) the cost ratio of the P-hybrid to the PS architecture. We consider here only the scenario of $\gamma = 1$ and homogeneous link costs (i.e. $lhc = 1$). Other scenarios such as $\gamma = \{0.1, 0\}$ and $lhc = 0.1$ have exhibited similar results. We set the download interval of the prefixes to $T_d = 10^4$ min (roughly equivalent to *one week*) and the number of videos to $K = 100$. We vary the storage capacity Cap of the set top-box between 100 and 1000 min of video. The popularities of the videos are Zipf distributed with $N_i = \frac{A}{i^{\alpha}}$. As we can observe from figure 3.10(a), when the local storage capacity of the client increases, the P-hybrid system cost reduces as compared to the PS system cost. This reduction exceeds 45% provided a storage capacity of $Cap = 1000$ min and a system with very few popular videos ($\alpha = 1$).

Figure 3.10(b) gives the optimal partitioning of the set-top box amongst videos for different values of A, α, and the storage capacity Cap. The larger the buffer space at the client, the larger the number of videos that share that buffer and the longer the length of the prefixes stored locally. Yet, it might seem surprising that the length of the prefix does not necessarily increase monotonically with the popularity N of the video. In fact, the open-loop scheme (suffix transmission) performs well for very popular videos. Thus, it may be optimal to reduce the prefix length of the most popular video in order to free a place for the other popular ones.

Figure 3.10(b) also shows that given the value of the parameter A, as α increases, the number of



(a) $C^{system}_{P-hybrid}/C^{system}_{PS}$



(b) Optimal prefi x length in the set-up box

Figure 3.10: $C^{system}_{P-hybrid}/C^{system}_{PS}$ and optimal prefix length in the set-up box for $\gamma = 1$ and $lhc = 1$.

videos that have their prefix stored at the client side decreases while the length of the prefix becomes longer. Indeed, for a given Zipf distribution ($N_i = \frac{A}{i^\alpha}$), the popularity of video i decreases as α increases. As we mentioned before, P-hybrid model reduces the system cost as compared to the PS model by storing locally the prefix of the most popular videos. Compared to $\alpha = 0.7$, for $\alpha = 1$, there are fewer popular videos in the system that should have their prefix stored in the set-top box. In addition, given the value of A, increasing α increases the efficiency of the P-hybrid model (figure 3.10(a)). Actually, the P-hybrid model becomes more cost efficient as the cost of the most popular videos increases relative to the total system cost, which is the case when α increases.

Conclusion

Allowing the client to store locally the prefix of some of the most popular videos can reduce significantly the system cost relative to the PS system cost. The scenarios that we considered show that:

- A set-top box with a reasonable storage capacity of 1000 min reduces the overall system cost by up to 30% for systems with many popular videos.
- The reduction in the system cost exceeds 45% for systems with relatively few popular videos and a large number of non-popular videos.

3.6 Conclusions and Outlook

3.6.1 Summary

In this chapter we showed that the PS-model can be used to evaluate many interesting extensions to the PS architecture. First of all, we studied the scenario where the prefix servers can not placed at any arbitrary level throughout the network. Instead, we allowed only leaf and root placement of the prefix servers. Under these assumptions, we compared the system cost to the optimal one obtained when the prefix servers can be placed at any level in the network. As a result,

- A non optimal placement of the prefix servers can dramatically increase the system cost; by up to 450% for leaf placement.

- Root placement outperforms leaf placement in the case where the network cost is cheap as compared to the server cost. In this case, the cost increase is relatively small, up to 60%.

- Root placement becomes very costly when the network cost is high relative to the server cost, and the increase in the system cost can exceed 600% in case of a reduced last-hop cost (i.e. $lhc = 0.1$).

We also evaluated the PS-model for the case of short videos such as video clips or clips for product promotions. Our results prove that it is always cost-optimal to divide the video into prefix and suffix in the case of very popular clips. Moreover, we extended the PS-model and looked at the following scenarios:

- **Provisioning:**
  We showed how the PS-model can be used to compute the cost-optimal for a system with a set $\mathcal{V}$ of videos. Given the popularity of each video, the PS-model determines the system resources required in terms of network bandwidth, server I/O bandwidth, and server storage to optimize the total system cost.

- **Assignment of Prefixes into Prefix Servers:**
  We studied how the PS-model adapts the prefix length and the prefix placement for a set $\mathcal{V}$ of videos when the amount of resources for the prefix servers is given, which is for instance the case when a new set of videos must be optimally placed. The cost increase due to predefined capabilities of the prefix servers is very low over a wide region of our model, less than 5%. Even when the popularity distribution of the videos differs a lot from the initial one used to determine the prefix server resources, the increase in the system cost remains low, 25-30%.

- **Satellite Transmission of the Suffix:**
  We evaluated the delivery cost of the video in the case where we transmit the suffix via satellite. Actually, when the cost for the satellite transmission is low relative to the cost for the terrestrial transmission, using satellite to transmit the suffix can reduce the system cost as compared to the PS-model by up to 80%.

- **Set-top Box at the Client Side:**
  We studied the evolution of the system cost in the case where it is possible to store at the client side the prefixes of some videos. We developed an analytical cost model (P-hybrid model) that allows us to share optimally the buffer space at the set-top box among the videos in the system. We found that storing locally the prefixes of the most popular videos can reduce efficiently the system cost as compared to the PS-model, by $30 - 45\%$.

The PS-model has allowed us to study different scenarios. However, several extensions are still possible that allow to make the overall model more realistic. The current model assumes that client requests for a single video are homogeneously distributed among all clients. A possible extension would consider the case where a particular video is more popular with a sub-group of the clients which would lead to heterogeneous request patterns. On the other hand, the current video distribution network has a very regular structure with all clients being at the same distance from the root. In contrast, a real distribution network most likely has not such a regular structure. While these extensions will clearly change the absolute cost values, we do not expect a change in the broad conclusions that we could draw using PS-model.

3.6.2 Outlook

In the system model we analyzed, we assumed that both, the suffix server and the prefix servers, are *dedicated* to the service. These assumptions hold for a “commercial” CDN. In recent years, a new paradigm called peer-to-peer (P2P) [55] has emerged where a particular machine can assume both roles, i.e. client and server at the same time. Popular P2P systems such as Gnutella [2], Napster [4], or KaZaa [3, 60] have been used by Millions of users to share digital content such as MP3 files. In addition, P2P architectures are very interesting for scalable content distribution. They offload all or at least the majority of the work for storage and distribution onto end-systems that are *not dedicated* to the purpose of content distribution and therefore, significantly reduce capital expenditures. P2P architectures are also inherently *self-scaling*: In case of a sudden increase in the number of requests, the number of clients that have received the content also increases, which in turn increases the total capacity of the P2P system to serve new clients. P2P systems are therefore much more suitable than centralized server/client systems to handle “flash crowds”.

The research community has made numerous proposals on how to use the P2P paradigm to perform a scalable video distribution [24, 14, 45]. In the next chapter we extend our work on VoD to P2P networks. We study how to provide an efficient VoD service in such an environment. Our contribution lies in a new pull-based approach (PBA) where, in contrast to existing approaches, we keep the server as simple as possible. Despite its simplicity, our approach achieves scalability, efficiency, and flexibility for additional optimizations.

Chapter 4

A Pull-Based Approach for an Efficient VoD Service in P2P Networks

4.1 Introduction

Nowadays, client hosts have significant storage and bandwidth power that could greatly reduce the capital expenditures of the service provider. Based on this observation, a new paradigm, called peer-to-peer (P2P), has emerged where a client assumes both roles, client and server at the same time. The P2P paradigm has been initially discussed for file sharing services (Napster, Kazaa, Gnutella, etc . . .). The central idea here is that files are stored at the clients side instead of at dedicated machines. When a client wants to access a given file, it first performs a lookup algorithm to locate a set of hosts that hold that file and then downloads the file. Many content lookup algorithms have been proposed such as, *Chord* [88], *Pastry* [79], and *TOPLUS* [38].

However, the use of P2P is not limited to file sharing services. P2P represents also an efficient infrastructure for content distribution. The most promising property of P2P distribution networks is that the responsibility of the content distribution 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; they download the content and upload it to other clients and so on. As a consequence, P2P networks can be an efficient solution for flash crowds: The larger the number of clients in the system, the larger the service capacity of the system and the better it scales and serves the content.

Video content distribution in P2P networks is currently receiving a lot of attention from the research community. Most of the existing proposals focus on how to achieve live streaming [30, 66, 21, 62, 25, 14, 27, 89, 93] and only few work deals with on-demand video distribution [85, 45]. These proposals advise to construct multicast trees to deliver the video to clients. However, in a dynamic environment like P2P networks[1], constructing and maintaining multicast trees make the server very complex. In this chapter we concentrate on the on-demand video distribution case and we show that we can achieve a good efficiency while keeping the server simple, i.e. no multicast trees are constructed.

4.1.1 Our Contribution

Tree-based approaches usually require the server to perform complex algorithms to construct and maintain the multicast trees. While a multicast tree seems to be intuitive and optimal for live streaming, VoD applications have the constraint that clients are asynchronous, which reduces somehow the efficiency of

[1]By dynamic nature we mean that clients may leave the network at any time.

multicast. In this chapter, our goal is to show that we can achieve both, a simple server algorithm (i.e. no multicast trees are constructed) and an efficient resources management. Our contribution here is a new pull-based approach, called PBA, for a scalable VoD service in P2P networks. In this approach, we do not intend to replace the unicast server/client framework; we rather extend this classical framework to serve more clients when the load on the server is high. The basic idea of PBA is quite simple. When a new client wishes to receive a video, the new client contacts first the server. If there is enough free bandwidth along the path to that new client, the server feeds the new client with the video. Otherwise, the server responds to the new client with a list of C_s servants (active clients) chosen at random. Any existing client that has received or is currently receiving the video can be a servant. The new client then searches for an *available servant* from which it downloads the video. An available servant is a candidate servant that has enough free upload capacity to serve the new client's request. So, PBA is not concerned with constructing multicast trees to serve a new client. In contrast, a new client performs locally a search for one available servant for the video. Moreover, leave events are handled locally at the clients side without involving the server (section 4.7).

We evaluate PBA and compare it to P^2Cast [45], a multicast tree-based approach. Our conclusion is that PBA does not only simplify the server, but it also consumes low network bandwidth, which allows a lot of clients to be serviced.
In its basic version, the list of servants is selected at random by the server. We also discuss a possible improvement to PBA: Instead of picking up C_s servants at random, the server performs a simple algorithm to choose the C_s closest ones to the new client. Choosing closest servants in terms of physical distance saves more bandwidth, which reduces the rejection probability of clients.

4.1.2 Organization of this Chapter

The rest of this chapter is organized as follows. Section 4.2 introduces the PBA model. In section 4.3 we overview the simulation settings. In sections 4.4 and 4.5 we evaluate extensively the performance of PBA through simulations. We also provide a comparison of PBA with P^2Cast. In section 4.6 we assess the gain in the system performance that we can achieve when we account for the physical distance between clients and their servants. Section 4.7 addresses how PBA can handle service disruptions and section 4.8 concludes the chapter.

4.2 PBA: A Pull-based Approach for VoD Service

Our model involves three entities: the server, the new client, and the servant:

4.2.1 The Server Task

PBA's key idea is to keep the server as simple as possible. When a new client wants to receive a video, it first contacts the server. Two scenarios are then possible:

- There is enough available bandwidth along the path between the server and the new client. In this case, the server delivers the video directly to the new client **via unicast**.

- There is not enough available bandwidth. In this case, the server responds to the new client with the IP addresses of C_s servants chosen at random. These C_s servants are selected amongst active clients in the network that have received or are currently receiving the video. The new client then tries to find one available servant from which it downloads the video and always via unicast.

Actually, estimating the available bandwidth between two end hosts in the Internet is not an easy task. Moreover, the available bandwidth of a connection might fluctuate over time. However, in this chapter we consider a static network. Our goal here is to show that as compared to a multicast tree-based approach, a pull-based approach simplifies the server and saves bandwidth resources. In addition, assuming a static network allows us to draw new insights while keeping simple the analysis. Thus, we assume that (i) Each connection has a transfer rate equal to the playback rate b [Mbps] of the video and (ii) A connection is set up (and can not be interrupted) in case the free bandwidth along the path of that connection is larger than or equal to b.

4.2.2 The New Client Task

As mentioned above, a new client first sends a request to the server to retrieve the video. If the available bandwidth along the path between the server and the new client is less than b, the server provides the new client with the IP addresses of C_s servants chosen at random. The new client then tries to improve this initial list of servants by accounting for local ones, i.e. servants that are located in its local area network. The motivation is that, retrieving the video from local clients saves bandwidth in the core of the network, which allows more clients to access the service. To discover local servants, the new client broadcasts locally a message. Each client that receives this message responds to the new client indicating its arrival time. We denote by C_s^l the number of local servants found. If C_s^l is greater than C_s, the new client replaces its initial list of servants by the C_s most recent local ones. On the other hand, if C_s^l is lower than C_s, the new client replaces the C_s^l oldest servants in its initial list by the C_s^l local ones. Afterwards, the new client searches for an available servant to download the video from. The new client contacts its servants starting with the local ones. If none of them is available, it contacts at random one of the non local ones. The new client keeps on contacting its servants until

- A servant accepts to serve it. In this case, the new client joins the network and sends a message to notify the server.
- None of the candidate servants accepts to serve the new client. In this case, the new client is rejected.

4.2.3 The Servant Task

When a servant receives a request for a video from a new client, the servant estimates the available bandwidth to the new client. If the available bandwidth is less than b, the servant rejects the request. In contrast, if the available bandwidth is larger than or equal to b, the servant unicasts immediately the video to the new client.

Even though our approach is centralized, we argue that it can scale to work well even when the load on the server is extremely high (e.g. flash crowds). In fact, the time needed for the server to process a join request is extremely short[2]. The server only needs to pick up C_s servants at random from its database. Choosing servants at random simplifies greatly the server and, at the same time, achieves a high connectivity among clients.
In PBA, the server maintains a database about all active clients in the system. This is not a lot as today's machines provide a large storage capacity. For instance, suppose there are one Million active clients in the system. The server needs to store the IP address (4 bytes) and the arrival time (4 bytes) of each client. This would not require more than 8 MB! The database is updated progressively as clients join and leave

[2]Leave events are handled locally at the clients side without involving the server.

the network. Each client must send an "alive" message to the server each T_a min. The value of T_a is a trade-off between the accuracy of the server's database and the messages overhead.

4.3 Performance Evaluation

Having introduced our model, we now evaluate its performance for various assumptions on the system parameters such as the network topology and the arrival rate of clients. We will also provide a comparison between PBA and P^2Cast that we describe briefly in the next section.

4.3.1 Overview of P^{2}Cast

P^2Cast applies *Controlled Multicast* to P2P systems. P^2Cast patches clients that arrive within the same interval of time into the same multicast stream, referred to as the base stream, that is transmitted over a multicast tree built on top of the clients. Within a session, the first client to arrive receives a complete base stream for the whole video from the server. When a new client arrives after the first one, we distinguish two cases: (i) The base stream that has been initiated for the first client is still active. In this case, the new client joins the multicast tree for the rest of the base stream. In addition, the new client receives a unicast patch for the part it missed in the base stream due to its late arrival. (ii) There is no active base stream. In this case, the server initiates a new stream and the above process is repeated for clients that arrive later. As in *Controlled Multicast*, the authors associate a threshold T_{Cast} (in min) to decide when to start a new base stream. The clients that share a given base stream form a session. Sessions are completely independents and there are no interactions between clients from different sessions. We will therefore focus on the transmission process within one single session.

Whenever a new session starts at time t_0, the first client receives the base stream from the server. A new client nc that requests the video within the interval of time $(t_0, t_0 + T_{Cast}]$ contacts peer P starting from the server[3]. Upon receiving the request, the peer P estimates its bandwidth $BW(P, nc)$ to nc. Meanwhile, P asks its children to estimate their bandwidth to nc. The child that has the largest bandwidth $BW(cmax, nc)$ to nc amongst all children is denoted by $cmax$. In case of tie, the closest child is chosen. P compares $BW(P, nc)$ to $BW(cmax, nc)$ and three scenarios are then possible:

- $BW(P, nc) > BW(cmax, nc)$ and $BW(P, nc) \geq 2b$. In this case, nc receives the patch directly from P and joins the multicast tree for the base stream as a child of P.

- $BW(P, nc) > BW(cmax, nc)$ and $2b > BW(P, nc) \geq b$. Since the patch has a larger priority as it is viewed before the base stream, P delivers the patch to nc and forwards the request for the base stream to its child $cmax$. $cmax$ then estimates its bandwidth to nc and asks also its children to estimate their bandwidth to nc and the same process as above is repeated.

- ($BW(P, nc) > BW(cmax, nc)$ and $BW(P, nc) < b$) or ($BW(P, nc) < BW(cmax, nc)$), P forwards the request to $cmax$ and the above process is repeated.

If the new client nc finds a servant for both, the patch and the base streams, nc gets access to the service. Otherwise, the new client is rejected.

4.3.2 Simulation Settings

In this study, we consider the case of a single video of length L min. The multiple videos case is left as a future work where the challenges would include how to decide which client to serve what

[3]In this thesis, we use the term peer only when we want to refer to both, clients and server at the same time.

video in order to optimally use the system resources. As we already mentioned, in P^2Cast, clients that share the same multicast stream form a session. Sessions are completely independents and there are no interactions between clients from different sessions. In contrast, PBA has no such constraint and a new client can retrieve the video from any other client in the network. To make a fair comparison between both approaches, we should take into account the session limitation of P^2Cast. Therefore, we introduce for PBA a threshold, denoted as T_{pba} (in min), to limit the interaction between clients. Suppose that client 1 has joined the network at time t_0. Client 1 can download the video only from active clients that arrived within $(t_0 - T_{pba}, t_0]$. As a consequence, this threshold serves to limit the number of candidate servants a new PBA client can contact.

In the following, we assume that clients request the video from the beginning to the end (no early departure) and that there are no service disruptions. In fact, service disruptions represent a main challenge for P2P systems. In section 4.7, we discuss how PBA can deal with such a scenario. However, in the performance evaluation we are interested in evaluating the bandwidth consumption and the percentage of rejected clients for PBA and P^2Cast. Therefore, we will assume no service disruptions and consequently, a client leaves the system only if (i) It has completely received the video and (ii) It is not serving any other client. Under the above assumptions, we can easily conclude that there is a maximum amount of time T_s (in min) a client can spend in the network:

$$T_s = \begin{cases} max(L, 2 \cdot T_{cast}) & \text{for } P^2Cast \\ L + T_{pba} & \text{for } PBA \end{cases} \tag{4.1}$$

In the above equation, the value $T_s = max(L, 2 \cdot T_{cast})$ for P^2Cast can be explained as follows. Consider a new session that starts at time t_0. Clients can join this session up to time $t_0 + T_{cast}$. Each client receives the base stream and an additional unicast patch in case it arrives after time t_0. The base stream is of length L min and will be completely delivered to all clients by time $t_0 + L$. The patch stream, on the other hand, can be up to T_{Cast} min. Clients that join the session as late as possible, i.e. at time $t_0 + T_{cast}$, will receive a patch of length T_{Cast} min until time $t_0 + 2 \cdot T_{cast}$. We can easily verify that, regardless of its arrival time to the session, each client will completely receive the video (the base and the patch streams) no later than time $max(t_0 + L, t_0 + 2 \cdot T_{cast})$. An extreme but possible scenario is that client 1 that joins the session as early as possible (i.e. at time t_0) serves the patch to a client 2 that joins the session as late as possible (i.e. at time $t_0 + T_{cast}$). In this case, and under the assumption that there are no service disruptions, client 1 must stay in the network until it completely delivers the patch to client 2 at time $t_0 + 2 \cdot T_{cast}$. This extreme scenario represents the case where client 1 stays in the network for a maximum amount of time equal to the whole duration of the session. And the value of T_s for P^2Cast is therefore $T_s = max(L, 2 \cdot T_{cast})$ min.

A similar analysis can be applied on PBA. In PBA, clients that arrive within T_{pba} min can interact with each other. A client that gets access to the service at time t_0 can serve new clients that arrive up to time $t_0 + T_{pba}$. Thus, it is possible that a client 1 starts delivering the video to a new client 2 that arrives at time $t_0 + T_{pba}$. In this case, client 1 must stay connected until it completely serves the video to client 2 at time $L + T_{pba}$ and consequently, $T_s = L + T_{pba}$ min for PBA.
The importance of the parameter T_s is that it helps deducing suitable values for the thresholds T_{pba} and T_{Cast} for the comparison: The values of T_{pba} and T_{Cast} are chosen in such a way that T_s is the same in both approaches.

We inspire our simulation settings from the evaluation of P^2Cast [45]. We use the network generator topology GT-ITM [95] to produce our network. The size of the network is a parameter that we study in this chapter. The network includes two levels, the *transit-domain* and the *stub-domain* levels (see figure 4.1 as an example). We assume that each stub-domain node is an abstraction of a local area network. However, we limit the number of streams inside each local network to L_{bw}. For instance, consider a local

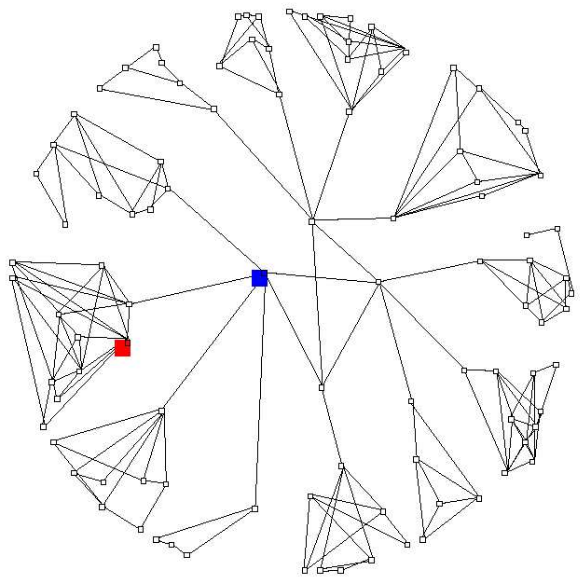


Figure 4.1: A 100 nodes network topology.

network LN for a stub-domain node. At any time, the maximum number of connections that have at least one of its two ends located in LN can not exceed L_{bw}. Each connection has a transfer rate of b, where b is the playback rate of the video. We set the capacity of each link between two transit-domain nodes (BW_{tt}) or between a transit-domain node and a stub-domain node (BW_{ts}) to $BW_{tt} = BW_{ts} = 20$. This means that, at the same time, such a link can support up to 20 streams each of rate b. For a link between two stub-domain nodes, we set its capacity to $BW_{ss} = 5$.

We assume that clients arrive to the network according to a Poisson process with a total aggregate rate λ clients/min. Clients connect only to stub-domain nodes, which all have the same probability to receive a new client.

In both approaches, PBA and P^2Cast, we account for the upload capacity C_{up} of clients. We consider two values of $C_{up} = \{2, 10\}$. The value $C_{up} = 2$ (respectively 10) means that an existing client can deliver at most two concurrent streams (respectively 10 streams) each at rate b.

In the performance evaluation, we are interested in (i) The percentage of rejected clients; (ii) The average number of hops the data traverse in the network. The term hop accounts for a link between any two nodes in the network; (iii) The system cost, which is expressed in units of b. The system cost is computed as the sum of the amount of bandwidth expended over all links in the network during an interval of time of length L min. In fact, in P2P distribution networks, end-hosts are not dedicated and therefore, the only important factor in the system cost is the network bandwidth spanned for the application; (iv) The link utilization expressed in units of b.

We start our evaluation with a basic scenario where:

- The network includes 100 nodes (figure 4.1) with four transit-domain nodes (the four nodes in the middle) and 96 stub-domain nodes.
- The server is placed in the transit domain, i.e. the shaded square in the middle in figure 4.1. We denote by S_p the server placement (S_p =transit-domain).
- We set T_{pba} to 10 min (i.e. 10% of the video length) for PBA. From equation 4.1 we obtain

$T_s = L + T_{pba} = 110$ min. Given that value of T_s, the value of T_{cast} for P^2Cast is $T_{cast} = 55$ min.

- The number of servants a client can contact is $C_s = 10$.
- The bandwidth capacity of each local area network is $L_{bw} = 50$.

We then extend our analysis to cover different scenarios where we change

- The values of T_{Cast} and T_{pba},
- The placement of the server S_p,
- The number of servants C_s a client can contact,
- The bandwidth capacity of the local area networks L_{bw}, and
- The size of the network.

Remaining parameters such as BW_{tt}, BW_{ts}, BW_{ss}, and L remain unchanged. All the results that we give are for one single video of length $L = 100$ min.

As defined in chapter 2, the parameter N stands for the popularity of the video, i.e. the average number of clients that request the video during an interval of time of length L min ($N = \lambda \cdot L$). In addition to this parameter N, we introduce here a new parameter N_a that is defined as follows. As explained before, some of the arriving clients get rejected. Thus, N_a represents the average number of admitted clients or clients that get access to the service during an interval of time of length L min.

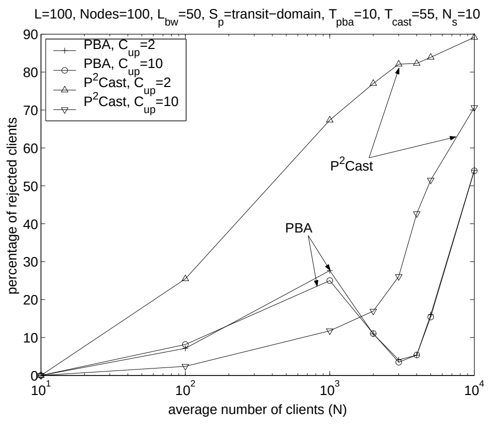
4.4 Basic Results

In this section we give a first set of results for the basic scenario that we just described. We run the simulation for a duration of 1000 min = 10 $\cdot$ L, where the first 100 min are used to warm up the system. Table 4.1 summarizes the parameters and their values used in this section.

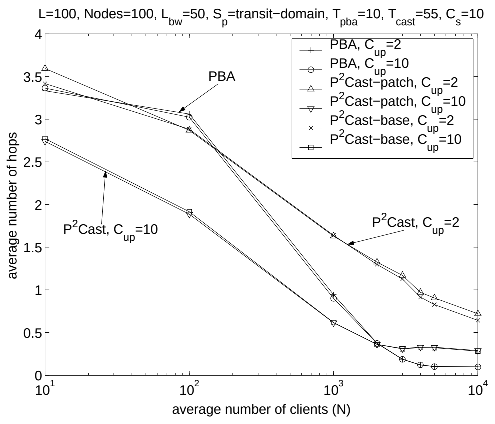
Parameters	**Values**
Simulation time	1000 min
Video length L	100 min
Number of nodes	100
Server placement S_p	transit-domain
λ	$\{0.1, 1, 10, 20, 30, 40, 50, 100\}$ clients/min
T_s	110 min
T_{cast}	55 min (55% of the video length)
T_{pba}	10 min (10% of the video length)
L_{bw}	50
BW_{tt}	20
BW_{ts}	20
BW_{ss}	5
C_{up}	$\{2, 10\}$
C_s	10

Table 4.1: Parameter values.

In figure 4.2(a), we plot the percentage of rejected clients for PBA and P^2Cast versus the video popularity N. Figure 4.2(a) shows many interesting features:



(a) percentage of rejected clients



(b) average number of hops

Figure 4.2: The percentage of rejected clients and the average number of hops as a function of the video popularity N, for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of candidate servants is $C_s = 10$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.

- As compared to P^2Cast, PBA reduces significantly the percentage of rejected clients when the video is highly popular. This reduction can reach 80%. Thus, our model is not only scalable in terms of processing incoming requests at the server side, but also in terms of network resources management.

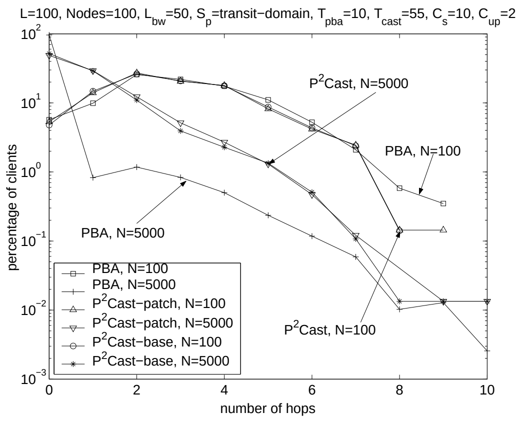
- In contrast to P^2Cast, PBA achieves a good efficiency with a reasonable demand on the client capabilities, i.e. $C_{up} = 2$.

As we can observe from figure 4.2(a), under a low arrival of clients ($N = 10$), the two approaches perform well and almost no clients are rejected. For P^2Cast, as N increases, the percentage of rejected clients increases monotonically. In addition, to achieve a reasonable performance, P^2Cast requires clients to have a large upload capacity (i.e. $C_{up} = 10$). In contrast, for PBA, the percentage of rejected clients does not increase monotonically with N. This is due to the threshold T_{pba} that we have introduced. T_{pba} limits the number of candidate servants for new clients. For instance, for $N = 100$ (i.e. 100 arriving clients per L), on average, 10 clients arrive within an interval of time $T_{pba} = 10$ min. Thus, few clients can collaborate with each other and candidate servants will be probably located far from the new client. Therefore, clients need to go far in the network to reach their servants. As a result, more bandwidth is consumed in the core of the network, which leads to congestion and clients rejection. When N exceeds 1000, the probability of finding a local servant increases and more clients are served locally. This saves bandwidth in the core of the network and allows to serve more clients. Consequently, the percentage of rejected clients starts decreasing for PBA. For $N > 5000$, the performance of both, PBA and P^2Cast deteriorates. This is mainly due to the limitation of the number of streams in the local networks ($L_{bw} = 50$ streams per local network).

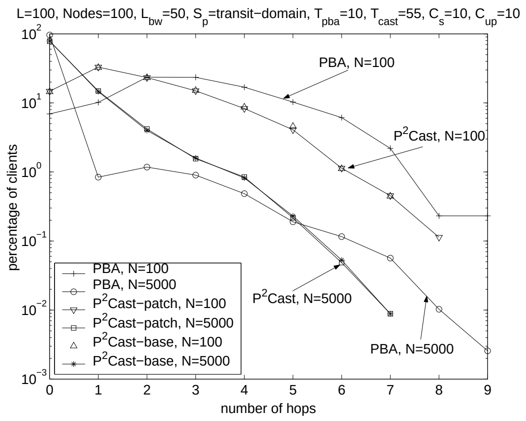
Figure 4.2(b) supports our intuition. This figure depicts the average distance (in terms of hops) between clients and their servants. Recall that, the term "hop" accounts for a link between any two nodes. Thereby, we do not account for the distance traveled by the data inside the local networks. So, the value $zero$ for the number of hops between a servant and a client means that both, the servant and the client are located in the same local network. As we can observe from figure 4.2(b), as N increases, the average number of hops decreases. For $N > 1000$, the average number of hops becomes smaller than 1, which means that the majority of clients are served by local servants.

With P^2Cast, the first client to arrive in the session receives only one stream, i.e. the complete base stream. Clients that join the session after the first one receive two streams, the patch and the base streams. In figure 4.2(b) and the next figures we plot results for each stream separately. If we give a closer look at figure 4.2(b), we see that the average number of hops is the same for both, the patch and the base streams. One possible reason is that, in most cases, client retrieves both streams from a same servant.

In figure 4.3, for each value of the number of hops, we plot the percentage of clients that are at that distance to their servants. In both approaches, as N increases, the percentage of clients served by close



(a) $C_{up} = 2$



(b) $C_{up} = 10$

Figure 4.3: The percentage of clients that are at the same distance to their servants for different values of the video popularity N, for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of candidate servants is $C_s = 10$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.

servants increases.

As a conclusion, at a high system load, PBA allows more clients to access the service as compared to P^2Cast. Let us now consider the total system cost for the two approaches. The system cost is computed as the sum of bandwidth consumed over all links in the network during an interval of time of length L. In figure 4.4 we draw the system cost for both approaches as a function of the number of admitted clients N_a. We denote by C_{cost}^{PBA} and $C_{cost}^{P^2Cast}$ the system costs of PBA and P^2Cast respectively. As we can observe from figure 4.4, PBA consumes less bandwidth than P^2Cast for large values of N_a. For moderate values of N_a (i.e. $N_a = 100$), P^2Cast slightly outperforms PBA. As mentioned above, this is due mainly to the threshold T_{pba} that we introduced for PBA to limit the interaction between clients. When N_a reaches 1000, C_{cost}^{PBA} decreases, which means that new clients always find local servants to download the video from and less bandwidth is consumed in the core of the network.

If we look at C_{cost}^{PBA} for $N_a > 5000$, we can notice that it increases again. The reason is that, limiting the bandwidth capacity of local networks to $L_{bw} = 50$ streams forces clients to connect to servants further away.

To confirm what we have seen so far, we plot the complementary cumulative distribution function $F^c(x)$ for the link utilization for $N = \{100, 5000\}$ and $C_{up} = \{2, 10\}$. $F^c(x) = 1 - F(x)$ is the probability that the link utilization *is larger* than x. For a moderate video popularity $N = 100$ (figure 4.5), PBA consumes more bandwidth than P^2Cast. The reason is that, due to the threshold T_{pba}, admitted clients in PBA have few candidate servants and therefore they retrieve the video from servants that are located far away in the network. As N increases, there are more candidate servants and PBA consumes less bandwidth than P^2Cast (figure 4.6).

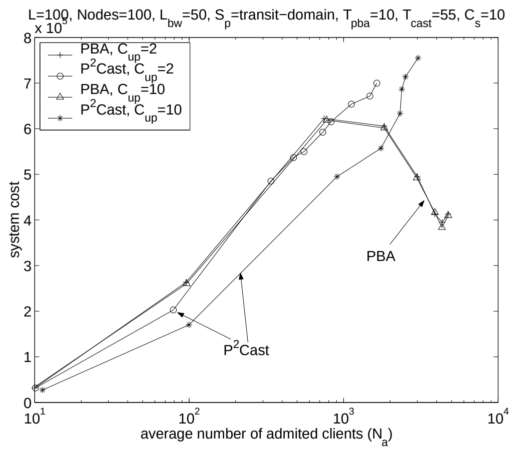
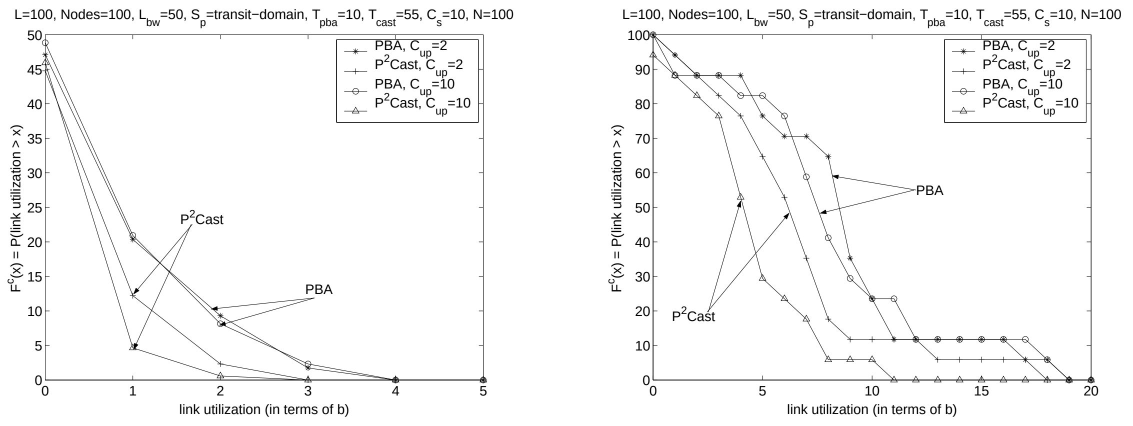


Figure 4.4: The system cost for PBA and P^2Cast versus the number of admitted clients N_a, for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of candidate servants is $C_s = 10$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.



(a) stub-domain links (b) transit-domain links

Figure 4.5: The probability $F^c(x)$ that the link utilization is larger than x, for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of candidate servants is $C_s = 10$, $T_{pba} = 10$ min, $T_{cast} = 55$ min, and $N = 100$.

4.4.1 Preliminary Conclusions

We evaluated the performance of PBA for a basic scenario where we considered (i) A network of 100 nodes, (ii) The bandwidth capacity of each local network is set to $L_{bw} = 50$, (iii) The server is placed in the transit-domain, (iv) The threshold values are $T_{pba} = 10$ min and $T_{cast} = 55$ min, and (v) The maximum number of candidate servants is set to $N_s = 10$. We also gave a comparison between PBA and P^2Cast. Our results for this scenario showed that

- PBA outperforms significantly P^2Cast for large values of the video popularity N. As compared to P^2Cast, PBA can reduce the percentage of rejected clients by up to 80% and the system cost by a factor of 2.

- Due to the threshold T_{pba} that we introduced, P^2Cast outperforms slightly PBA for moderate

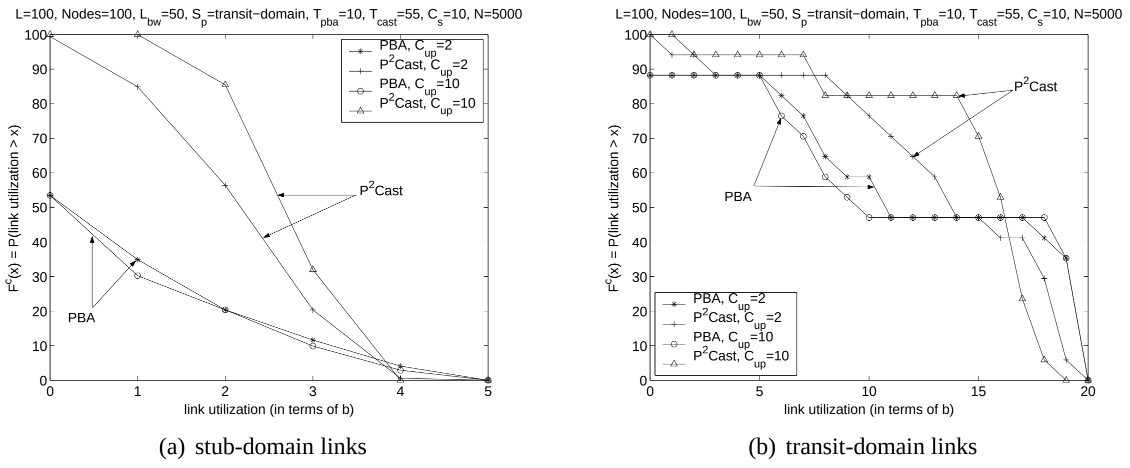


Figure 4.6: The probability $F^c(x)$ that the link utilization is larger than x, for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of candidate servants is $C_s = 10$, $T_{pba} = 10$ min, $T_{cast} = 55$ min, and $N = 5000$.

values of the number of clients $N = 100$.

- In contrast to P^2Cast, PBA achieves a good efficiency with a reasonable demand on the client capabilities, i.e. $C_{up} = 2$.

Outlook

This basic scenario validated our intuition that we can achieve both, a simple server algorithm and an efficient system resources management. Yet, we still need to validate more our results. In the next sections we continue our performance evaluation and investigate how PBA and P^2Cast behave when we change separately the parameters presented below:

- The values of T_{Cast} and T_{pba} (section 4.5.1),
- The placement of the server (section 4.5.2),
- The number of servants C_s a client can contact (section 4.5.3),
- The bandwidth capacity of the local area networks L_{bw} (section 4.5.4), and
- The network topology (section 4.5.5).

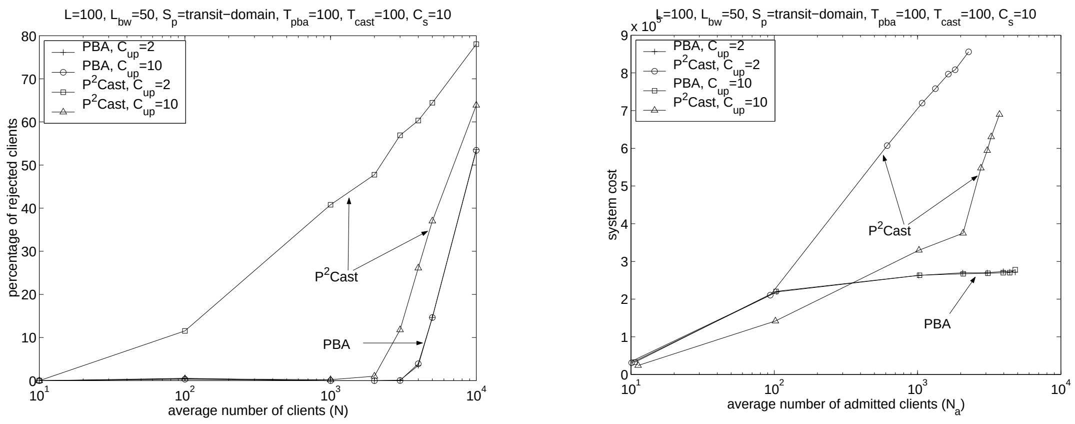
We also discuss how PBA can be very flexible in the sense that many features can be added to the system. One possible improvement is to allow the server to choose the closest servants to the new client. The rational behind is that choosing closest servants in terms of physical distance saves network bandwidth and allows more clients to access the service. Finally, we describe in section 4.7 how PBA can be extended to scale in real environment where clients can leave the system at any moment.

4.5 Heterogeneous Results

4.5.1 Impact of the Threshold Values

In P^2Cast, clients need to store the patch part of the video which is at most of T_{cast} min of length. In contrast, PBA requires clients to store only T_{pba} min of the video; a PBA client will be delivering at most the last T_{pba} min that it already displayed. In the basic scenario that we considered, we set $T_{cast} = 55$ min and $T_{pba} = 10$ min. We now consider an extreme scenario where clients in both approaches store the same amount of data of L min. In this case, the values of T_{pba} and T_{Cast} become $T_{cast} = T_{pba} = L$ min and consequently, $T_s = 2 \cdot L$ min. Except T_{pba}, T_{Cast}, and T_s, all parameter values presented in table 4.1 remain valid.

Figure 4.7(a) depicts the percentage of rejected clients as a function of the number of clients N while figure 4.7(b) draws the system cost versus the number of admitted clients N_a. These two figures confirm



(a) percentage of rejected clients (b) system cost

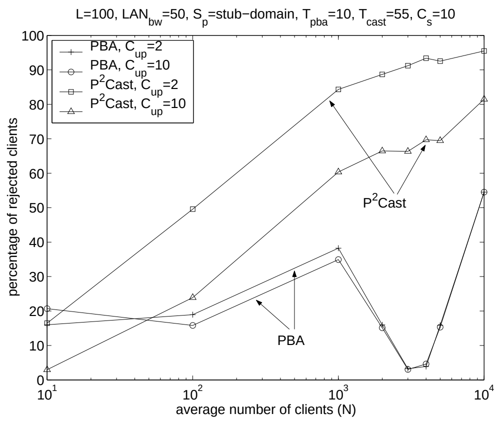
Figure 4.7: The percentage of rejected clients as a function of N and the system cost as a function of N_a for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of servants is $C_s = 10$, and $T_{pba} = T_{cast} = 100$ min.

what we obtained in the basic scenario with $T_{cast} = 55$ min and $T_{pba} = 10$ min. In addition, we can clearly see that the performance of both, PBA and P^2Cast, improves with the threshold value. Indeed, in both approaches, the larger the value of the threshold, the larger the number of clients that can interact one with another, which improves the performance of the system.

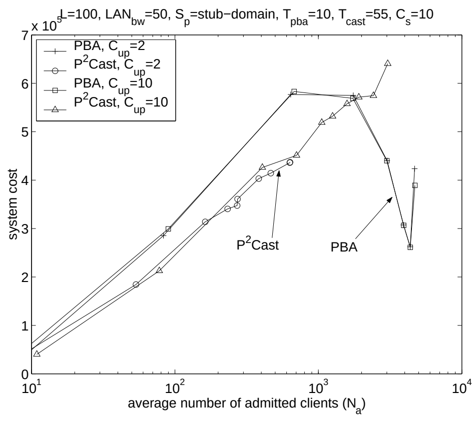
4.5.2 Impact of the Server Placement

So far, we have placed the server in the transit-domain level. In this section we study the impact of the server placement on the performance of PBA and P^2Cast. For this purpose, we move the server to the stub-domain level (the shaded point in the stub domain in figure 4.1). By doing so, we reduce the upload capacity of the server, which intuitively would reduce the performance of the system. In this scenario, except the server placement, all other parameter values given in table 4.1 remain unchanged.

In figure 4.8(a) we plot the percentage of rejected clients as a function of the video popularity N. As compared to the basic scenario (figure 4.2(a)), figure 4.8(a) shows that at a low system load (i.e. $N = 10$), the server placement has mainly an impact on PBA; for $N = 10$, there are on average 10 arrivals per $L = 100$ min, which makes one arrival each T_{pba} of 10 min. So, most probably, a new client will have no candidate servants that arrived to the network within T_{pba} min. As a result, the new client can retrieve the video only from the server. As its upload capacity has been reduced, the server can service less clients and therefore, more clients are rejected. In contrast, P^2Cast has a larger threshold which allows more clients to interact one with the other, which explains why it has a relatively low rejection rate. As N increases, the impact of the server placement on PBA decreases while it increases on P^2Cast; the performance of P^2Cast decreases faster with N as compared to the case where the server is placed in the transit-domain.



(a) percentage of rejected clients



(b) system cost

Figure 4.8: The percentage of rejected clients as a function of N and the system cost as a function of N_a for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the stub-domain, the number of servants is $C_s = 10$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.

If we compare the system costs in figures 4.4 and 4.8(b), we do not notice a big change in the overall behavior. Due to the threshold T_{pba}, P^2Cast consumes less bandwidth than PBA at low and moderate values of N. In contrast, PBA performs better for large values of N where the impact of T_{pba} is low.

4.5.3 Impact of the Number of servants

In PBA, when the server is fully using its upload capacity, it provides a new client with a set of C_s servants. In P^2Cast, on the other hand, when a new client contacts the server, the server performs an algorithm to decide whether to serve the client or to forward its request to one of its children. The same process is repeated at each level of the tree until finding an available servant for each stream (i.e. the patch and the base streams). Hence, the parameter C_s in P^2Cast limits the number of times the request is forward down in the tree, which in turn limits to C_s the depth of the multicast tree.

We now study two values of $C_s = \{5, 30\}$. Other parameters in table 4.1 will have the same values. In figure 4.9 we plot the percentage of rejected clients against the video popularity N for $C_s = \{5, 30\}$. This figure shows a very interesting result: In most cases, a client in the two approaches finds an available servant within 5 tries. Only for $N = 1000$, PBA requires a larger number of candidate servants, i.e. $C_s = 30$. As concerns the system cost, figure 4.10 reveals the same tendency for both approaches as in the basic scenario (figure 4.4).

4.5.4 Impact of the Bandwidth Capacity of Local Networks

To make the results more realistic, we have introduced the parameter L_{bw} that represents the bandwidth capacity inside a local network. For sack of simplicity, we assume that all local networks have the same capacity. Up to now, we have considered a value of $L_{bw} = 50$, which means that a local network can hold up to 50 streams each at rate b. In this section we evaluate the impact of this parameter on the performance of PBA and P^2Cast. We choose an extremely low value, $L_{bw} = 5$. However, we believe that in practice, the value of L_{bw} is even larger than 50.

It is obvious that the performance of both approaches depends on the parameter L_{bw}. This parameter limits the number of simultaneous clients in each local network. In figure 4.11(a) we plot the percentage of rejected clients versus the system load N while figure 4.11(b) shows the system cost as a function of

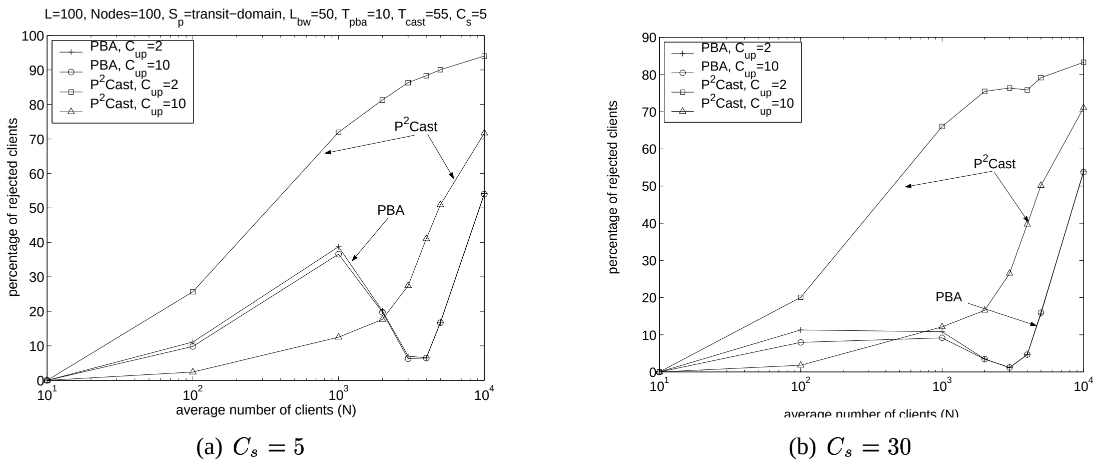


Figure 4.9: The percentage of rejected clients as a function of the system load N, for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of servants is $C_s = \{5, 30\}$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.

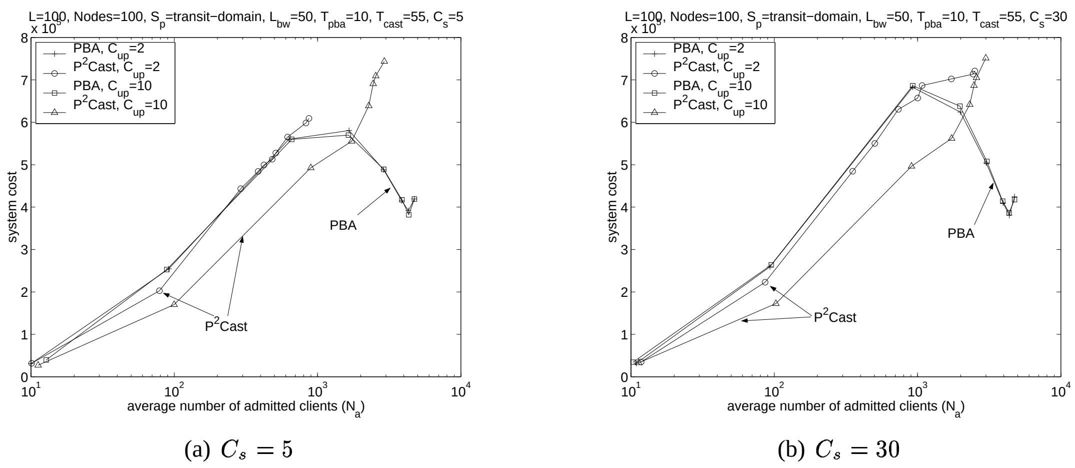
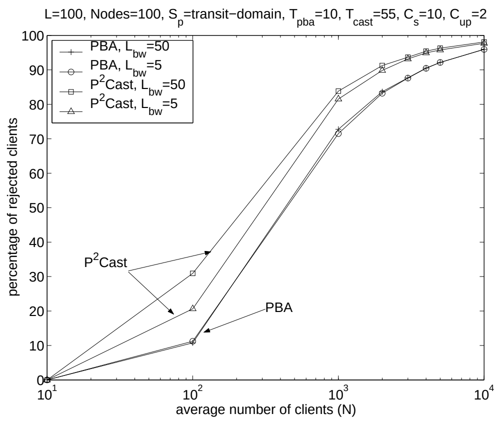


Figure 4.10: The system cost as a function of the number of admitted clients N_a for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of servants is $C_s = \{5, 30\}$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.

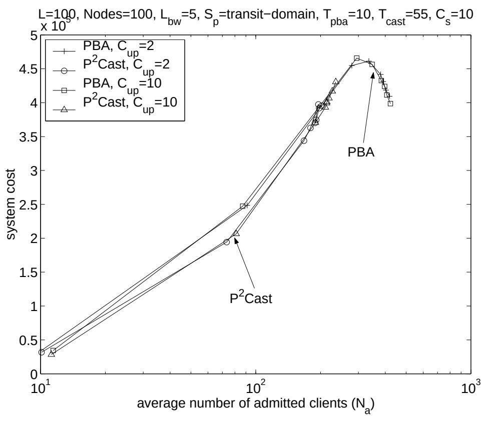
the number of admitted clients N_a. Up to $N = 100$, on average, there is almost one client in each local network and thus, L_{bw} does not intervene strongly. As N increases from 100 to 1000, the percentage of rejected clients increases significantly. In fact, when $N \geq 1000$, the main limitation of the system is the bandwidth capacity of local networks. Even though, PBA is still more advantageous than P^2Cast (figure 4.11(a)).

4.5.5 Impact of the Network Topology

We now investigate the impact of the network topology on the performance of both approaches. We consider a larger network of 1000 nodes (figure not given). As in the basic scenario, we obtained the network using the generator topology GT-ITM. The network consists of one transit-domain with 20 nodes and 49 stub-domains. Each transit-domain node holds on average 7 stub-domains, each of which includes on average 7 nodes, which makes an overall of 980 stub-domain nodes. This network provides



(a) clients rejection, $L_{bw} = \{5, 50\}$, $C_{up} = 2$



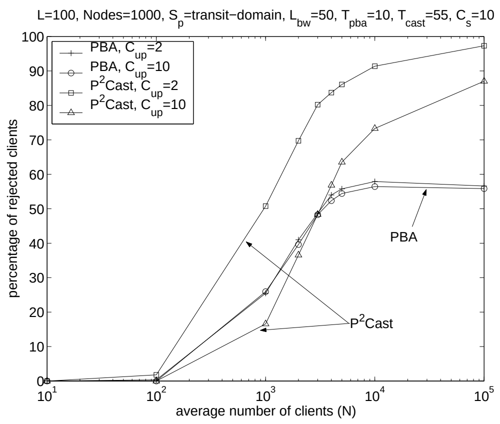
(b) system cost, $L_{bw} = 5$, $C_{up} = \{2, 10\}$

Figure 4.11: The percentage of rejected clients as a function of N and the system cost as a function of N_a for a network of 100 nodes, the server is placed in the transit-domain, the number of servants is $C_s = 10$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.

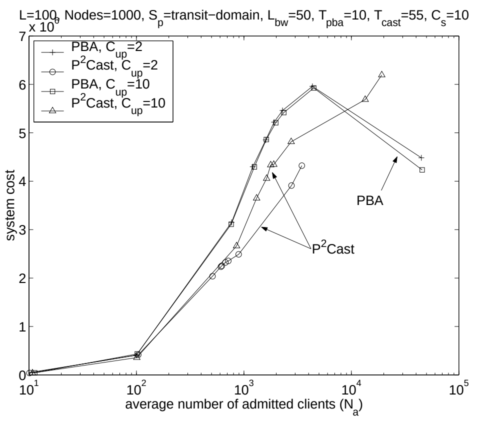
a high connectivity at the transit-domain level (there is a link between each two transit-domain nodes). In contrast, there is a lower connectivity at stub-domain levels. In a same stub-domain, there is a link between two nodes with a probability of 0.5.

In this scenario, except the number of nodes, all other parameter values given in table 4.1 remain valid. Note that we also extended the interval of values of the video popularity to $N = 10^5$.

As we can observe from figure 4.12, the performance of PBA and P^2Cast has improved for $N = 100$. A larger network means more resources and therefore, more clients can be served. We can also



(a) percentage of rejected clients



(b) system cost

Figure 4.12: The percentage of rejected clients as a function of N and the system cost as a function of N_a for a network of 1000 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of servants is $C_s = 10$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.

notice that for PBA, increasing the size of the network by 10 order of magnitudes has moved the value at which the percentage of rejected client decreases by the same order, from $N = 1000$ to $N = 10^4$ (figure 4.2(a) versus figure 4.12). The reason is that, in PBA, the percentage of rejected clients starts decreasing when a new client finds most often a local servant from which it downloads the video. As the size of the network has increased from 100 to 1000 nodes, for the same value of N, the probability of having a local servant decreases.

4.6 CSF: A Possible Improvement to PBA

In the initial version of PBA, if the estimated bandwidth along the path from the server to the new client is less than b, the server picks up at random the IP addresses of C_s servants and sends them to the new client. In this section, we evaluate an improved version of PBA in case the server is able to account for the physical distance between servants and clients. This means that, instead of choosing C_s servants at random, the server chooses the C_s closest ones to the new client. We refer to this approach as CSF (i.e. Closest Servant First). If two servants are at the same distance to the new client, the most recent one is chosen first.

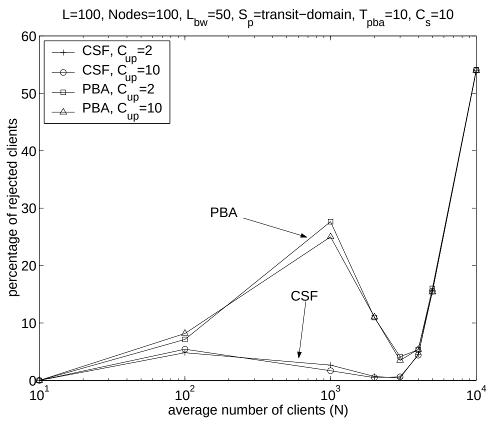
4.6.1 Finding Close Servants

Finding nearby clients in the Internet has been addressed recently and there already exist approaches that implement such a feature [38, 61, 65, 75, 83, 56]. For example, in *TOPLUS* [38], the authors regroup clients in clusters based on their IP addresses. They do so by using information from BGP routing tables and the prefixes of well-known networks (such as LANs or ISPs). The BGP information serve to regroup clients in coarse-clusters while the LANs or the ISPs information help to refine these clusters.
Thus, to find closest servants, the server can simply perform an algorithm based on the above mentioned solutions. Another interesting idea would be to perform the selection procedure of the servants completely at the client side. The server provides a new client with a quite large set of servants and always chosen at random. Upon receiving its list, the new client probes at random some of these servants using the tool *King* [43] (*King* is a very interesting tool that allows to estimate the delay between the DNS of two clients with high precision). Amongst the probed servants, the one that achieves the lowest delay is considered as the closest one; close servants are more likely to achieve low delay. While it is obvious that such a solution does not provide exact locality information, we believe that it can achieve good results without complicating the server.

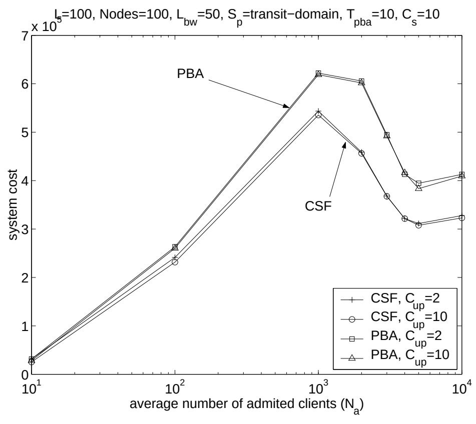
4.6.2 Results

We compare PBA to CSF to figure out the gain that can be achieved if (exact) locality information are available. In figure 4.13(a) we draw the percentage of rejected clients for both approaches versus the video popularity N. As we can observe from figure 4.13(a), CSF outperforms significantly PBA for large values of N (i.e. $N = 1000$). Indeed, in CSF, a new client always contacts a close servant. In contrast, in PBA, if the new client finds no local servant, it picks one candidate at random. This locality property of CSF allows it to outperform PBA for $N = 1000$. In fact, for $N = 1000$ arrivals per the video length L, there are 100 arriving clients within a $T_{pba}$of 10 min. These 100 clients will be distributed over 96 stub-domain nodes. This gives on average 1 client per stub-domain node. Thus, a new client finds with a large probability a close servant but not a local one. This explains why CSF outperforms PBA for such a value of N. In contrast, for $N = 100$, this locality property has no main impact as candidate servants are few (i.e. $\simeq$ 10 servants distributed over 96 nodes) and most probably located far away from the new client. Hence, choosing the closest servant or one at random give the same result. When N increases from 100 to 1000, with a high probability, a new client will find a close servant but not a local one. While this would increase the probability of picking up at random a close servant for new clients with PBA, CSF takes a full advantage of these close servants. As a result, the percentage of rejected clients decreases for CSF while it keeps on increasing for PBA. When N becomes larger ($N > 1000$), the probability of having a local servant increases and more clients will be served locally, which saves bandwidth in the core of the network and allows to serve more clients. Consequently, the percentage of rejected clients starts decreasing for PBA and remains decreasing for CSF. For high values of N ($N \geq 3000$), a new client always finds a local servant and as a consequence, both PBA and CSF have the same performance. For $N \geq 5000$, the performance of both approaches deteriorates due to limiting the bandwidth capacity of local area networks to $L_{bw} = 50$ streams.

Figure 4.13(b) depicts the system cost for each of the two approaches as a function of the number of admitted clients N_a. Figure 4.13(b) confirms our intuition, i.e. accounting for the physical distance between servants and clients allows to save bandwidth and reduces the system cost.



(a) percentage of rejected clients



(b) system cost

Figure 4.13: The percentage of rejected clients and system cost for both, PBA and CSF, for a network of 100 nodes, $L_{bw} = 50$, the server is placed in the transit-domain, the number of candidate servants is $C_s = 10$, $T_{pba} = 10$ min, and $T_{cast} = 55$ min.

4.7 Service Disruptions

A main challenge for P2P systems is how to deal with service disruptions. When a client leaves the network, all its descendents will stop receiving the stream until they find a new servant. Also the network state changes dynamically and the available bandwidth between two end hosts might fluctuate over time. This has for sure a severe impact on the service the system offers to its clients. PBA can be easily extended to handle these issues. The key idea is to allow clients to contact multiple servants instead of a single one. Clients perform locally a greedy algorithm to download different portions of the video from k servants at the same time. We can inspire from previous work [77, 73] to orchestrate the synchronization between the different download connections.

When a client receives no packet from one of its servants during a given interval of time, the client considers that the servant has left the network. It then drops the connection to that servant and sets up a new one with a new servant from its list. Furthermore, when the client departure rate is high, the number of servants a client maintains in its list C_s^{actual} may drop significantly. For this reason, each client sets a threshold C_s^{th}. When C_s^{actual} decreases lower than C_s^{th}, the client asks some of its active servants to provide it with the IP addresses of $(C_s - C_s^{actual})$ servants. So, leave events are handled locally at the client side without involving the server.

Note that the idea of using parallel download to retrieve the content has been introduced a few years ago by Rodriguez et al. [78]. Today, many of recent approaches like *BitTorrent* [26] rely on this idea to improve the throughput of clients. Parallel download helps distributing the load over the network. In addition, having multiple servants with a total aggregate download rate larger than b (the playback rate of the video) allows clients to quickly fill their buffer. Thereby, when a servant leaves the network, its children should have enough data in their buffer to find a new servant and consequently, experience no interruptions while playing out the video. Parallel download also makes clients less sensitives to the bandwidth fluctuation in the network since the parallel download algorithms always post more requests to the servants that offer the best response time and thus smooth out the load. We leave the quantitative evaluation of this parallel download approach as future work.

4.8 Conclusions and Outlook

4.8.1 Summary

In this chapter, we studied the use of the P2P paradigm for video content distribution. While previous approaches build multicast trees to distribute the video to clients, we believe that such solutions make the server quite complex. Our solution is a new pull-based approach that, in contrast to previous ones, keeps the server very simple and at the same time, achieves an efficient management of network resources.

The basic idea of our approach is very simple. When a new client wishes to receive a video, it first contacts the server. If there is enough free bandwidth along the path to the new client, the server transmits the video to the new client. Otherwise, the server responds to the new client with a list of C_s servants. These servants are clients that have received or are currently receiving the video. Then, the new client searches for an available servant from which it downloads the video.

We conducted intensive simulations to evaluate the performance of our model. We also provided a comparison between PBA and P^2Cast, a multicast tree-based approach proposed previously. Our results showed that as compared to P^2Cast, PBA not only simplifies the server, but it also consumes significantly less network bandwidth and allows more clients to access the service. We first gave results for a basic scenario where we considered (i) A network of 100 nodes, (ii) The bandwidth capacity of each local network is set to $L_{bw} = 50$, (iii) The server is placed in the transit-domain, (iv) The threshold values are $T_{pba} = 10$ min and $T_{cast} = 55$ min, (v) The maximum number of candidate servants is set to $C_s = 10$. A key result is that

- PBA outperforms significantly P^2Cast for large values of the video popularity N. As compared to P^2Cast, PBA can reduce the percentage of rejected clients by up to 80% and the system cost by a factor of 2.

- Due to the threshold T_{pba} that we introduced, P^2Cast outperforms slightly PBA for moderate values of the number of clients. e.g. $N = 100$.

- In contrast to P^2Cast, PBA achieves a good efficiency with a reasonable demand on the client capabilities, i.e. $C_{up} = 2$.

To validate our results in depth, we investigated further scenarios where we changed separately the following parameters (i) The values of T_{Cast} and T_{pba} , (ii) The placement of the server, (iii) The number of servants C_s a client can contact, (iv) The bandwidth capacity of the local area networks L_{bw}, and (v) The network topology. The results that we obtained confirmed the previous ones and exhibited the same tendency for both approaches.

We then showed that our model is highly flexible in the sense that many features can be added to the system. One possible extension was to allow the server to account for the physical distance between clients and their servants. We demonstrated that such a feature reduces the bandwidth consumed in the core of the network and allows the system to serve more clients.

Note that in the performance evaluation phase, we assumed a static network and no service disruptions. Our goal was to reveal that we can achieve a high efficiency without constructing multicast trees. However, we also discussed how PBA can deal with real scenarios where the network conditions change dynamically over time and clients can leave the network at any time. We identified the basic principle, i.e. the use of parallel download to retrieve the video from multiple servants simultaneously. Downloading different portions of the video from multiple servants improves the throughput of clients. Moreover, parallel download algorithms make clients less sensitive to the servant departures and bandwidth fluctuations in the network.

4.8.2 Outlook

In this chapter we focused on video content distribution in P2P networks. Another important related problem is the file replication. Nowadays, file replication using P2P has become a prominent service and a very significant source of Internet traffic.

Existing approaches for file replication in P2P networks can be largely classified into tree-based and mesh-based approaches, according to the way clients are organized in the network. In the next chapter, we first review the scaling behavior of both kinds of approaches. We prove that mesh approaches are not only simple and robust, but they also outperform tree ones. We then focus on mesh approaches and present a complete analysis where we isolate the main factors that guide the performance of these approaches.

Chapter 5

Mesh Approaches for File Replication in P2P Networks

5.1 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 [7] 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 [46].
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 chapter has two main goals. The first one is to compare tree-based and mesh-based approaches. Biersack et al. [18] 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 chapter 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. [94] 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 chapter 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]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.

5.1.1 Our Contribution

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

- **Peer selection strategy:** which among our neighboring peers will we actively trade with, i.e. serve chunks to or request chunks from? In this chapter, 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 [18]. 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 chapter 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 [21, 26, 84, 94] 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 chapter 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.

5.1.2 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 5.1 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 5.1: Definition of the parameters used in this chapter.

5.1.3 Organization of this Chapter

The rest of this chapter is structured as follows. In section 5.2 we introduce the basic design of our two architectures, *Least Missing* and *Most Missing*. Section 5.3 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 5.4. Section 5.5 describes the experimental setup and the simulations results are given in sections 5.6 and 5.7. We discuss open questions in section 5.8 and we conclude this chapter in section 5.9.

5.2 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.

5.2.1 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* [50].

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 chapter, 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 [82].

- *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.

5.2.2 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.

5.2.3 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 5.3.2).

Similarly, one could think of maintaining multiple download connections to improve the throughput of clients. Previous work by Gkantsidis et al. [40] has clearly showed that, in a large deployment of parallel download, more than 5 concurrent download connections does not benefit the clients. In this chapter 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.

5.3 Summary of Tree Approaches

Biersack et al. [18] 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.

5.3.1 *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 5.1). Basing on the above analysis, one could easily verify that the number of served clients within t rounds is given by:

$$\begin{aligned} N_{Linear}(|\mathcal{C}|, t) &= t + |\mathcal{C}| \cdot t(t-1)/2 \\ &\sim |\mathcal{C}| \cdot t^2 \end{aligned} \tag{5.1}$$

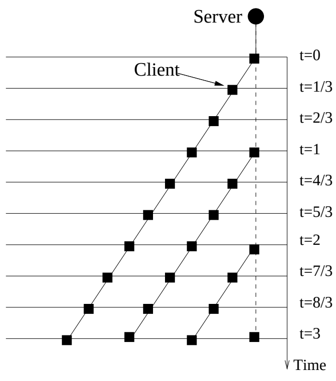


Figure 5.1: The evolution of the *Linear* architecture with time. The number of chunks is set to $|\mathcal{C}| = 3$. The black circle represents the server while the black squares represent the clients.

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. (5.1) they derive the time needed (expressed in number of rounds) to serve N clients as follows:

$$\begin{aligned} T_{Linear}(|\mathcal{C}|, N) &= \frac{(|\mathcal{C}| - 2) + \sqrt{(|\mathcal{C}| - 2)^2 + 8 \cdot N \cdot |\mathcal{C}|}}{2 \cdot |\mathcal{C}|} \\ &\sim \frac{|\mathcal{C}| + \sqrt{|\mathcal{C}|^2 + 8 \cdot N \cdot |\mathcal{C}|}}{2 \cdot |\mathcal{C}|} \end{aligned} \tag{5.2}$$

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 5.2 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 *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 $N = 10^6$.

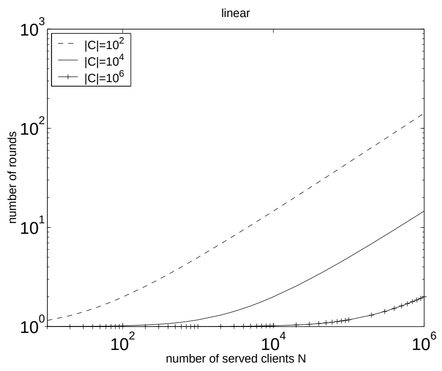


Figure 5.2: The service time for *Linear* versus the number of clients N for different values of the number of chunks, $|\mathcal{C}| = \{10^2, 10^4, 10^6\}$.

5.3.2 *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 5.3).

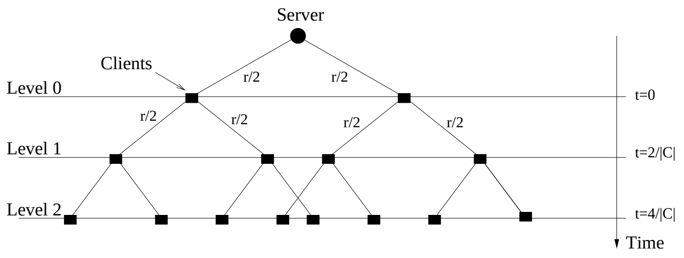


Figure 5.3: The evolution of the $Tree^k$ architecture with time for $k = 2$. The client needs two units of time to receive the whole file at rate $\frac{r}{2}$. The black circle denotes the server while the black squares denote the clients.

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:

$$N_{Tree}(|\mathcal{C}|, k, t) \sim k^{\lfloor \frac{(t-k)\cdot|\mathcal{C}|}{k} \rfloor + 1} \tag{5.3}$$

Eq.(5.3) 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:

$$T_{Tree}(|\mathcal{C}|, k, t) = k + \lfloor \log_k \frac{N}{k} \rfloor \cdot \frac{k}{|\mathcal{C}|} \tag{5.4}$$

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 $Tree^k$.

5.3.3 *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 5.4 depicts the basic idea of $PTree^k$ for $k = 2$. In $PTree^k$, each client helps in distributing

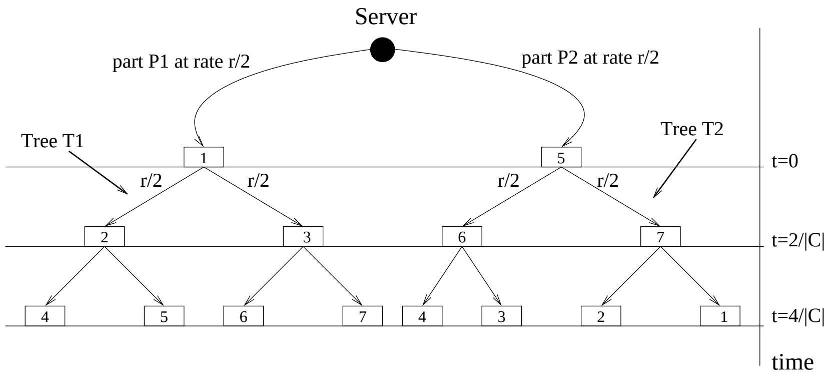


Figure 5.4: The evolution of $PTree^k$ with time. The file is divided into two parts ($k = 2$).

$\frac{1}{k}$ of the file (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. [21] 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

$$T_{PTree}(|\mathcal{C}|, k, N) = 1 + \lfloor \log_k N \rfloor \cdot \frac{k}{|\mathcal{C}|} \tag{5.5}$$

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:

$$\begin{aligned} N_{PTree}(|\mathcal{C}|,k,t) &= \frac{k^{(\lfloor (t-1)\cdot\frac{|\mathcal{C}|}{k}\rfloor+1)}-1}{k-1} \\ &\sim k^{(\lfloor (t-1)\cdot\frac{|\mathcal{C}|}{k}\rfloor)} \end{aligned} \tag{5.6}$$

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 5.2 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 them active most of the time.

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 5.2: Summary of the scaling behavior of *Linear*, $Tree^k$, and $PTree^k$.

5.4 Mesh approaches versus Tree approaches

As stated in the introduction, one of the two goals of this chapter 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.

5.4.1 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 5.5 draws the time at which clients start receiving the file. This above analysis

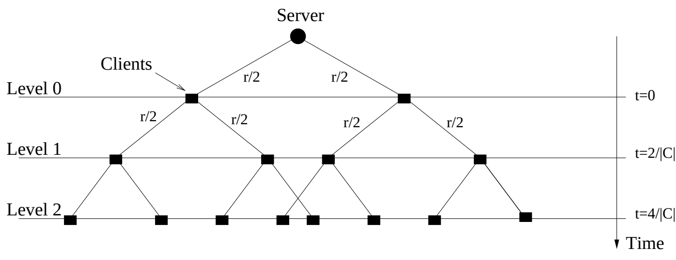


Figure 5.5: The evolution of the number of served clients with the *Least Missing* architecture. All N clients arrive to the system at time $t = 0$. Each peer has indegree $P_{in} = 1$ and outdegree $P_{out} = 2$. We assume homogeneous upload/download capacities $S_{up} = C_{up} = C_{down} = b$.

can 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*).

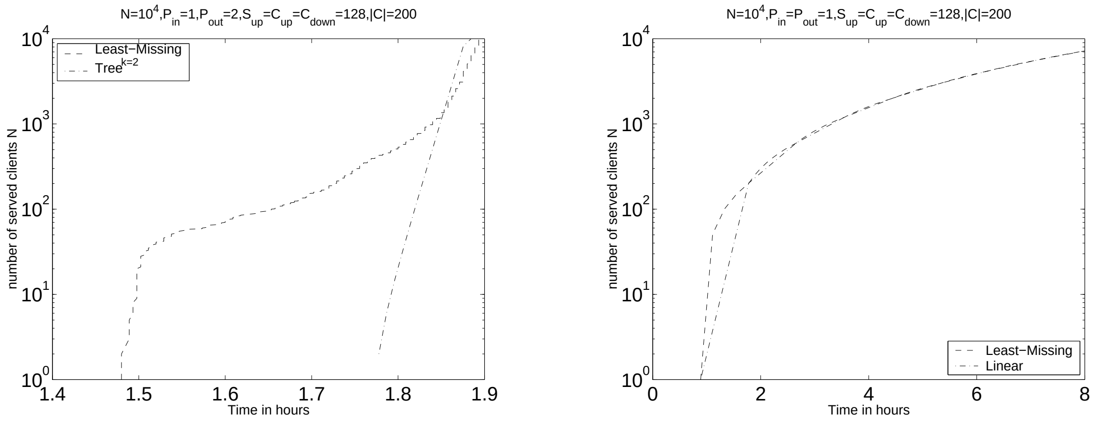
Results

To validate our analysis, we compare in figure 5.6 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 5.1 and 5.3 in sections 5.3.1 and 5.3.2. Figure 5.6 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 5.6(a)). Figure 5.6(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

[2]Details about the simulator are given in section 5.5

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 5.6(b)). As we can observe from figure 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*

Figure 5.6: 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.

5.4.2 *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 5.3. In figure 5.7

Table 5.3: 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

we expose the number of served clients against time. This figure supports our intuition. *Most Missing*

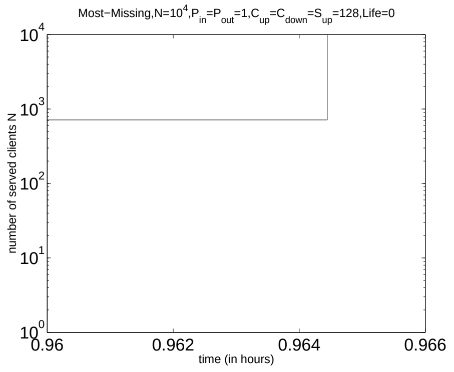


Figure 5.7: The number of served clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$).We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

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 5.3, 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 5.8 we present a snapshot of the download progress as seen by clients after 45 min of the beginning of the 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 5.9(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 5.9(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 5.9(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 5.9(b)). To explain the quick replication of chunks with *Most Missing*, we proceed as follows. For the parameter values displayed in table 5.3, one round is

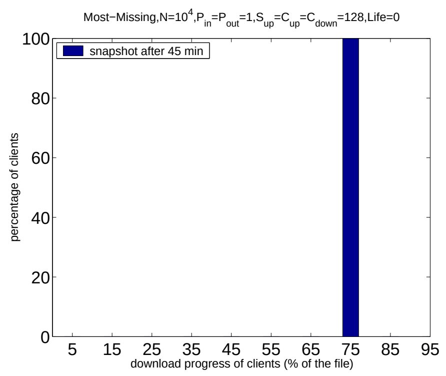
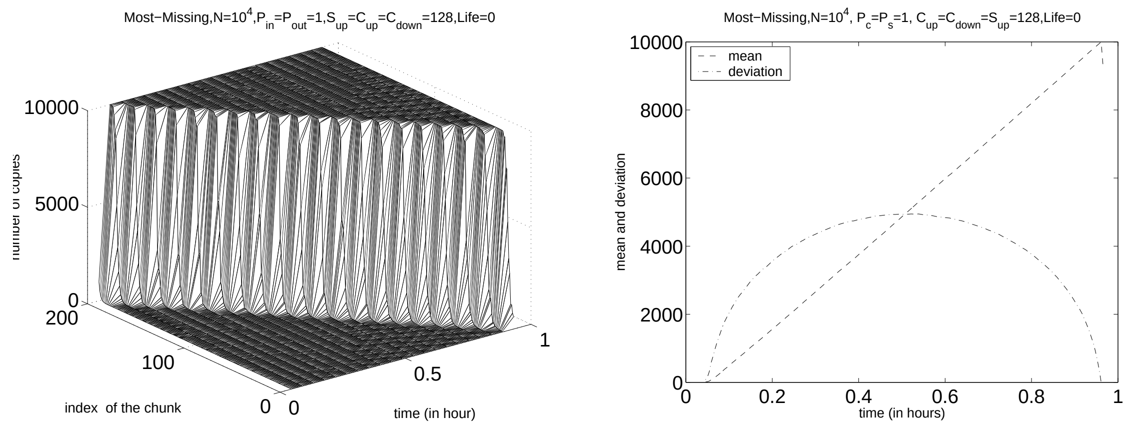


Figure 5.8: Snapshot of the download progress of clients for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).



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

Figure 5.9: 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 ($Life = 0$).

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, 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 5.10 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

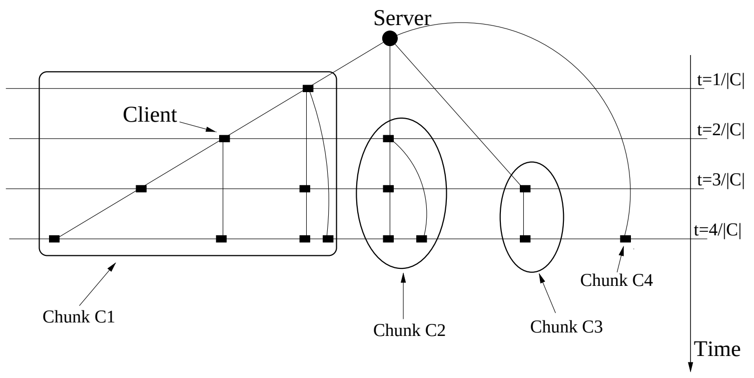


Figure 5.10: The theoretical evolution of the number of copies for the first four chunks with time for *Most Missing*. All N clients arrive to the system at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = r$. Clients disconnect as soon as they receive the file (*Life* $= 0$).

$t = \frac{2}{|\mathcal{C}|}$, there are 2 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 5.9(a) shows that the growth in the number of copies for subsequent chunks is as high as during the start-up period.

In figure 5.11, we plot the simulation result for the evolution of the number of chunks versus time. We show only 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.

5.4.3 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

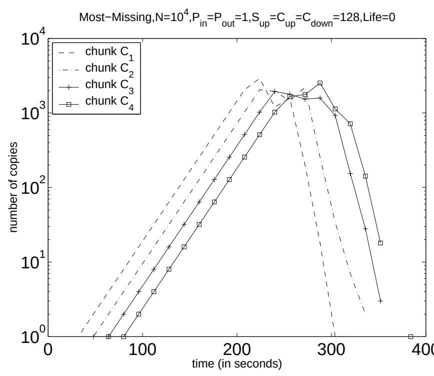


Figure 5.11: The simulation result for the evolution of the number of copies of chunks with time. We consider only the first four chunks injected by the server. All N clients arrive to the system at time $t = 0$. Each peer serves one single client at a time. We assume homogeneous upload and download 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 ($Life = 0$).

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.

5.5 Experimental Setup

For the purpose of evaluating our two cooperative architectures, we made use of the simulator developed by Felber et al. [36]. The simulator models various types of P2P networks and allows us to observe step-by-step the distribution of large files among all peers in the system according to several metrics.

5.5.1 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.

5.5.2 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.

5.5.3 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 5.4 summarizes the values of the main parameters used in our simulations.

Parameter	**Value**
Chunk size	256 KB
Number of chunk $\lvert\mathcal{C}\rvert$	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 5.4: Parameters used in the 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.
- *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.

5.6 Simulation Results: Instantaneous Arrival of Clients

5.6.1 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 5.4.2 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 5.12 compares *Least Missing* to *Most Missing* for the parameter values given in table 5.5. This figure exhibits that, overall, *Most Missing*

Table 5.5: 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

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 5.12(b) we can notice that *Least Missing* optimizes the service time of the first few clients, which is also clear from figure 5.13. In figure 5.13, 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 $t = \frac{1}{|\mathcal{C}|}$, 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

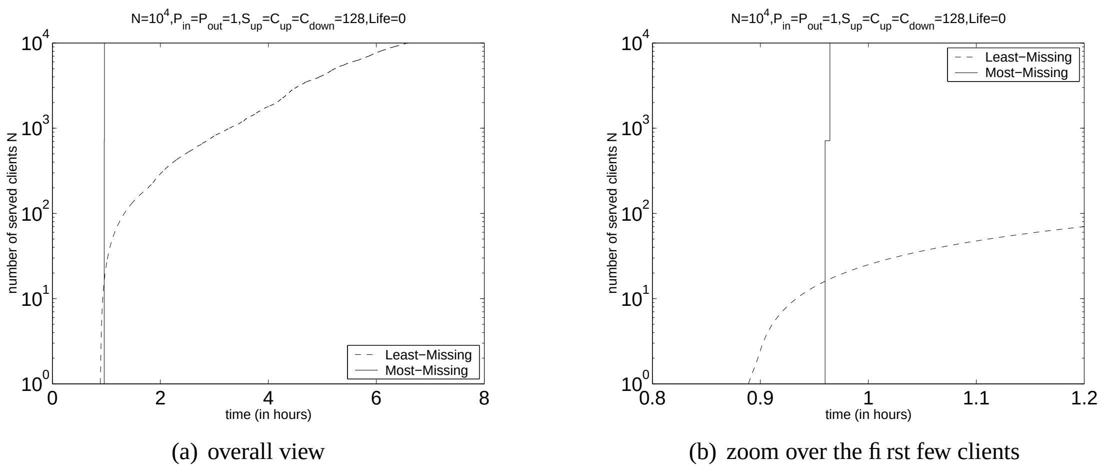


Figure 5.12: The number of served clients against time for *Least Missing* and *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

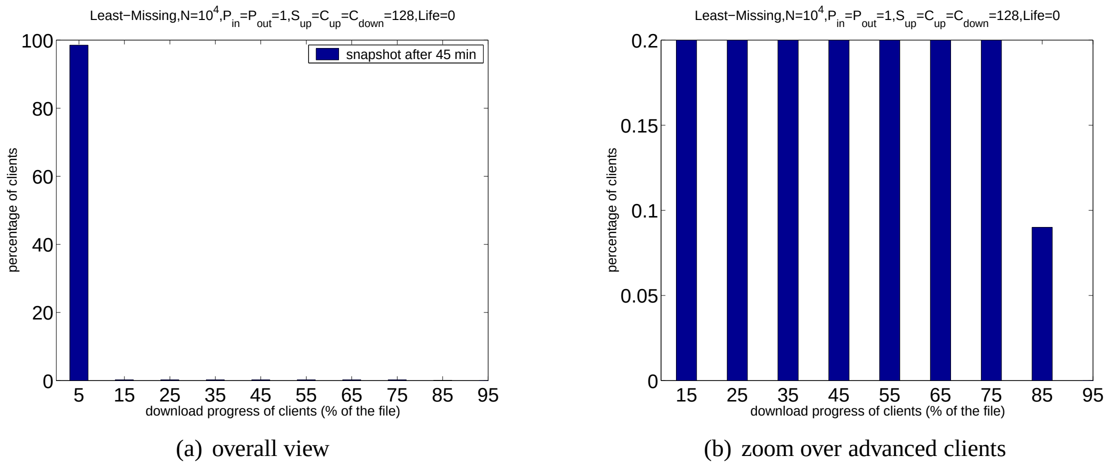


Figure 5.13: Snapshot of the download progress of clients for *Least Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

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 5.14, we can notice that the number of copies of the chunks grows linearly in time.

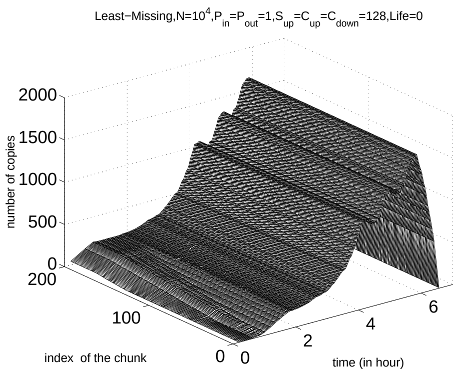


Figure 5.14: The chunks distribution over time for *Least Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients 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.

5.6.2 Impact of the Indegree/Outdegree of Peers

Previous work by Yang et al. [94] 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 5.6).

Table 5.6: 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 5.15. The results shown in this figure are in accordance with former ones: Clients download the file quickly and

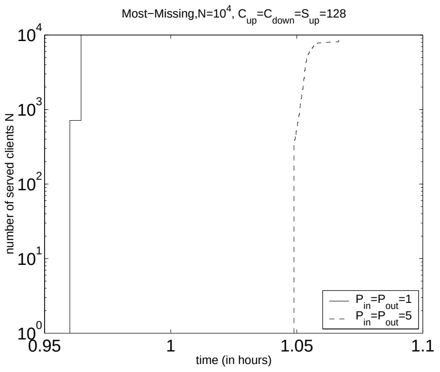


Figure 5.15: The number of completed clients against time for *Most Missing*. All N clients arrive at time $t = 0$. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

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 [94]: 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 5.16 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

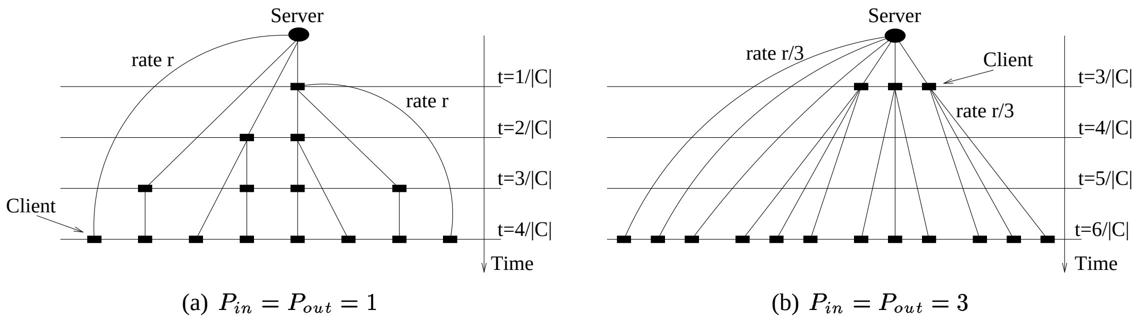


Figure 5.16: The evolution of the number of copies of one single chunk in *Most Missing*. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = r$.

$P_{in} = P_{out} = 3$. From figure 5.16, 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 5.17). As a result, we obtain

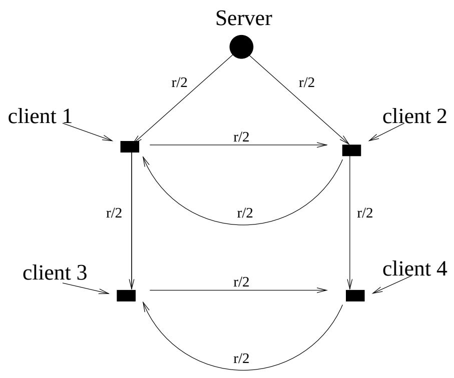


Figure 5.17: The theoretical evolution of *Least Missing* for $P_{in} = P_{out} = 2$. We focus only on the first four clients in the system.

two chains 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 5.18 plots the number of completed clients versus time. As we can 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 5.19, 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 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

[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.

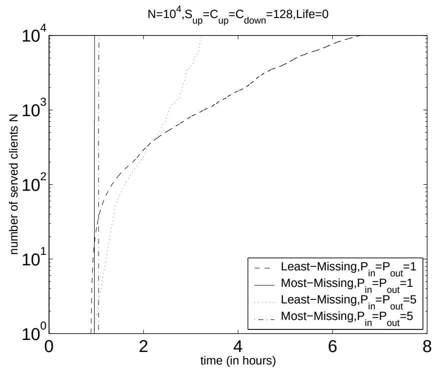
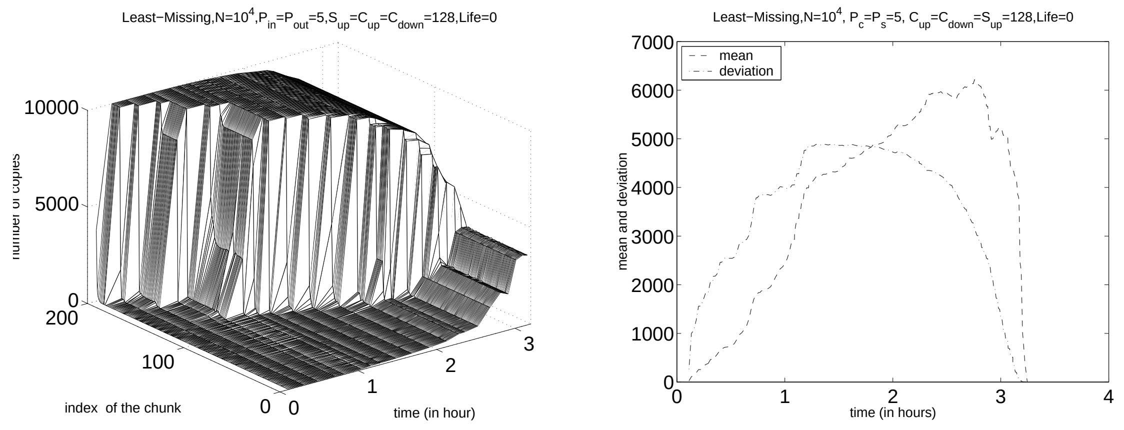


Figure 5.18: The number of completed clients against time. All N clients arrive at time $t = 0$. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).



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

Figure 5.19: 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* $= 0$).

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 5.20.

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.

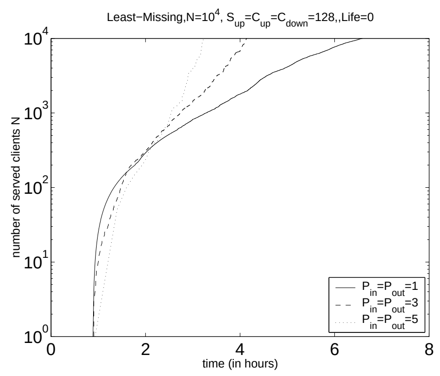


Figure 5.20: The number of completed clients against time for *Least Missing*. All N clients arrive at time $t = 0$. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

5.6.3 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 5.7.

Table 5.7: 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 5.21).

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 5.22 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 5.23, 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.

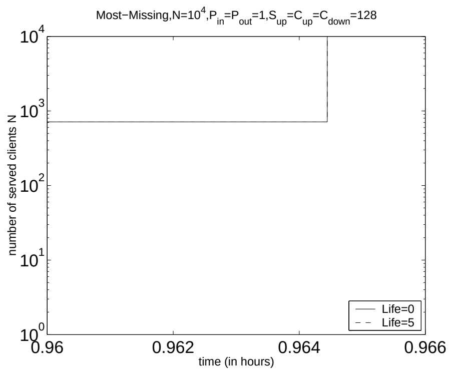


Figure 5.21: The number of completed clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connections ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps.

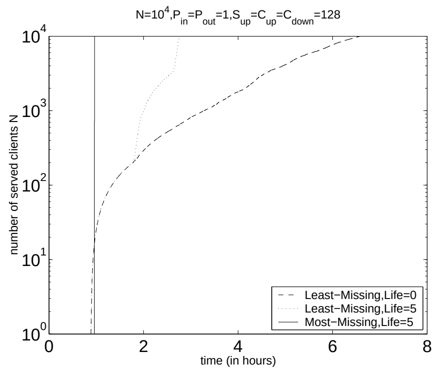


Figure 5.22: The number of completed clients against time. All N clients arrive at time $t = 0$. Each peer maintains at most 1 download and 1 upload connections ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps.

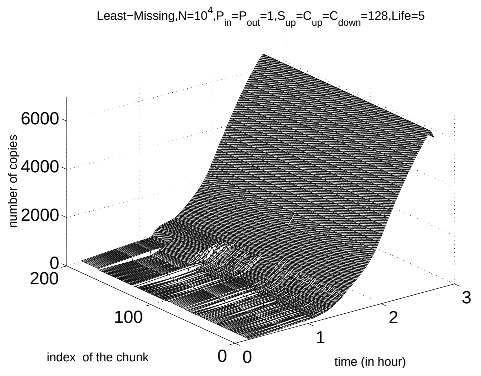


Figure 5.23: The chunks distribution over time for *Least Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most 1 download and 1 upload connections ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients stay online 5 additional min after they have received the file (*Life* $= 5$).

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.

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 5.8 outlines the remaining parameter values.

Table 5.8: 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 5.24 is completely different. Figure 5.24 shows two batches of clients: The 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 5.25). This above behavior of *Most Missing* is not really intuitive. 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

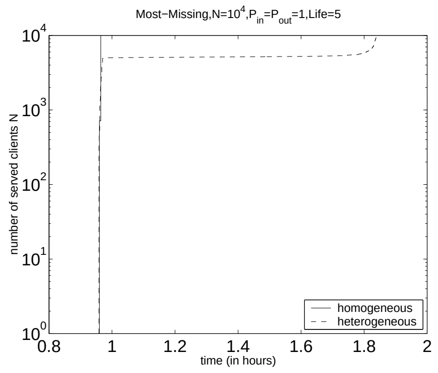


Figure 5.24: The number of completed clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). The upload and download capacities are $S_{up} = 1024$ Kbps, $C_{up} = 128/64$ Kbps, and $C_{down} = 512/128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

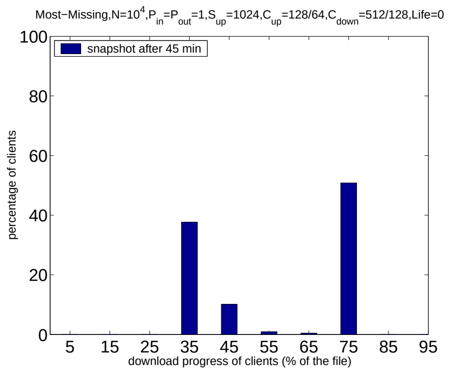


Figure 5.25: The download progress of clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). The upload and download capacities are $S_{up} = 1024$ Kbps, $C_{up} = 128/64$ Kbps, and $C_{down} = 512/128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

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 5.26, 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.

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 5.27 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

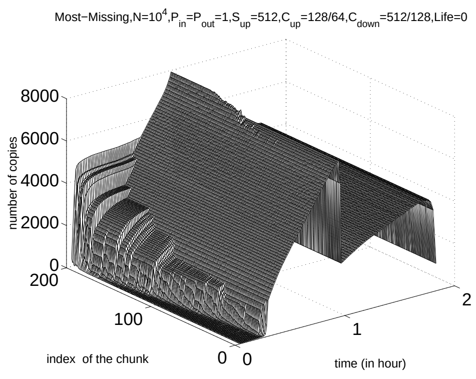


Figure 5.26: The chunks distribution over time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). The upload and download capacities are $S_{up} = 1024$ Kbps, $C_{up} = 128/64$ Kbps, and $C_{down} = 512/128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

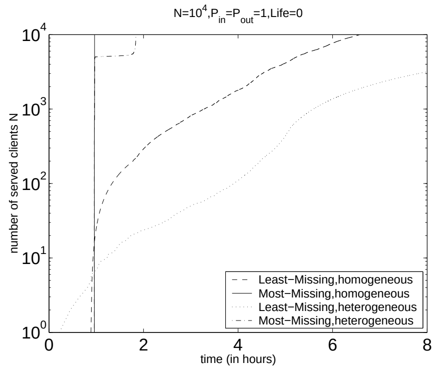
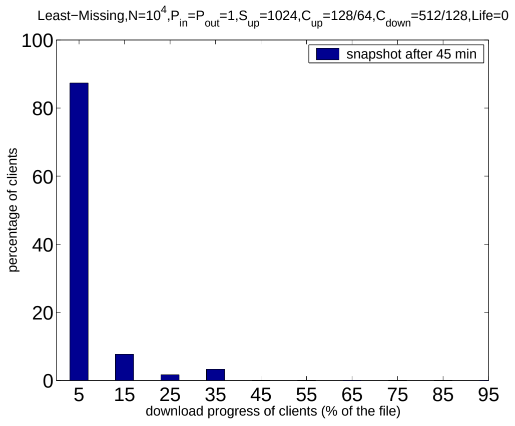


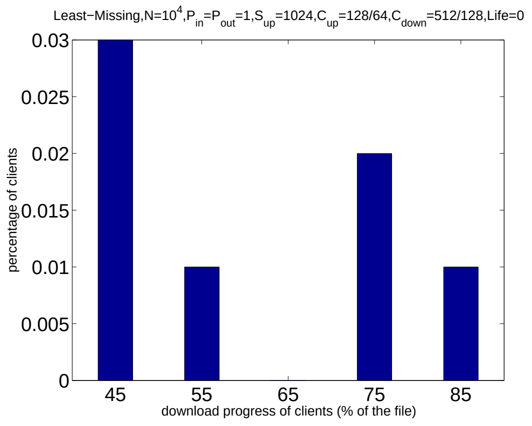
Figure 5.27: The number of completed clients against time. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). Clients disconnect as soon as they complete their download (*Life* $= 0$).

damaged by the low upload capacity of other clients. If we take a closer look at figure 5.27, 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 5.28).

What is new for *Least Missing* is that this scenario reveals a higher number of chunks in the system (figure 5.29) as compared to the basic scenario with homogeneous bandwidth capacities ($S_{up} = C_{up} = C_{down} = 128$ Kbps, figure 5.14). 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.

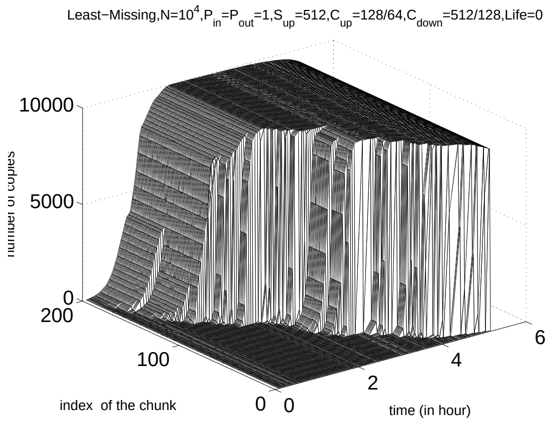


(a) overall view

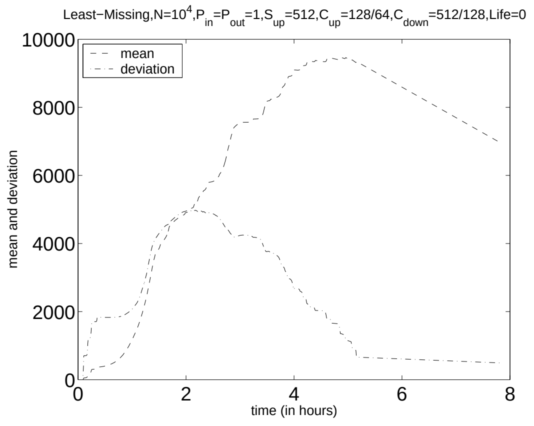


(b) zoom over advanced clients

Figure 5.28: 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* $= 0$).



(a) Number of chunks vs time and chunk index



(b) Mean and deviation vs time

Figure 5.29: 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* $= 0$).

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.

5.6.5 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 5.9.

Table 5.9: 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

Most Missing

We can notice the first impact of the chunk selection strategy in figure 5.30 where the number of served clients with *Most Missing* has completely a different scaling tendency. This figure shows that around

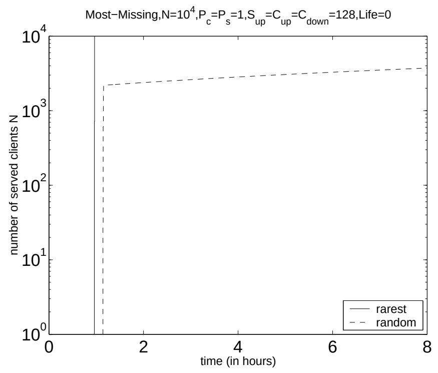
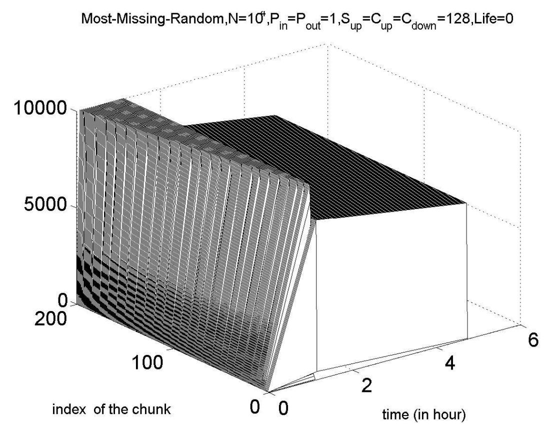
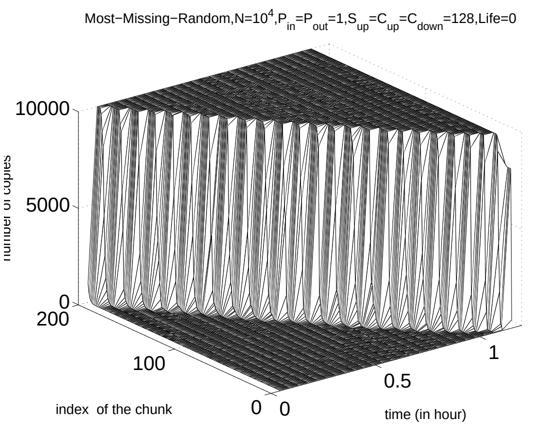


Figure 5.30: The number of completed clients against time for *Most Missing*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

2000 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 $\frac{1}{|\mathcal{C}|}$ unit of time, which is equivalent to 16 seconds under the parameter values of table 5.9. To explain this behavior of *Most Missing Random*, we graph the chunks distribution over time in figure 5.31. 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. As a result, one client completes each 16 seconds and, instead of **3472 seconds** to serve 10^4



(a) Overall seen



(b) Zoom over the first hour

Figure 5.31: 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* $= 0$).

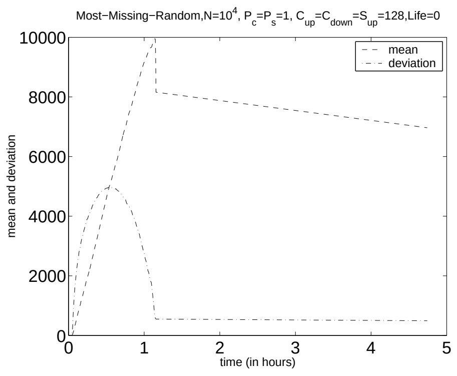


Figure 5.32: The mean and deviation of the number of chunks against time for *Most Missing Random*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

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 5.33 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 5.34) 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 $\frac{1}{|\mathcal{C}|}$ 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

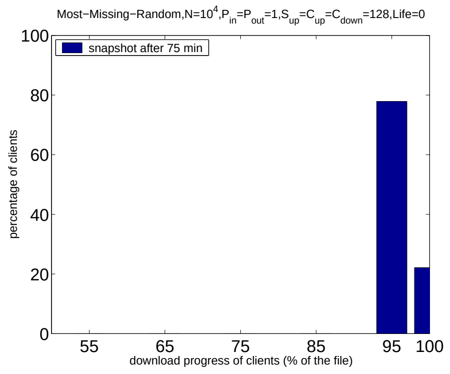


Figure 5.33: The download progress of clients against time for *Most Missing Random*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$.

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 $\frac{3}{|\mathcal{C}|}$, there are no new

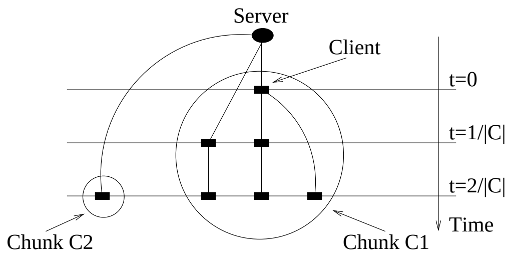


Figure 5.34: The distribution of the chunks over the different clients during the first $\frac{2}{|\mathcal{C}|}$ unit of time in *Most Missing Random*. We assume that the number of clients is $N = 7$ and the number of chunks is $|\mathcal{C}| = 2$.

comers and existing clients can start exchanging 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 ($Life = 0$). 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 5.35, choosing the appropriate chunk to be scheduled is also a key performance for the *Least Missing* policy. When

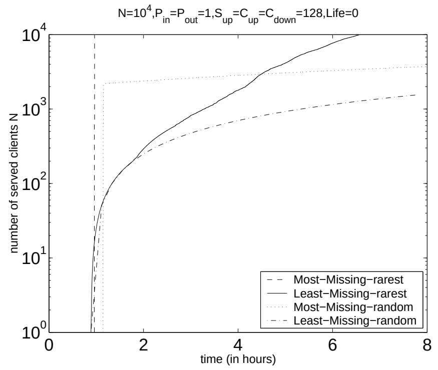
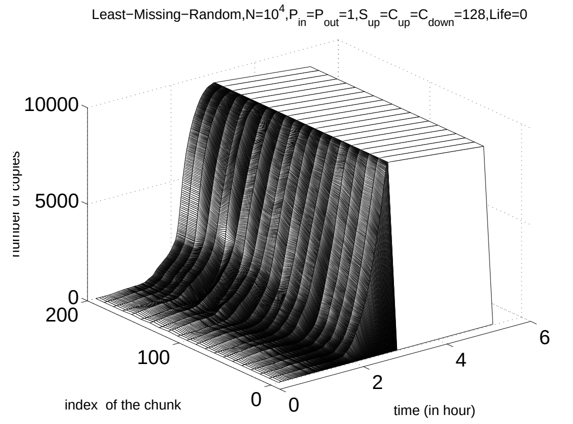
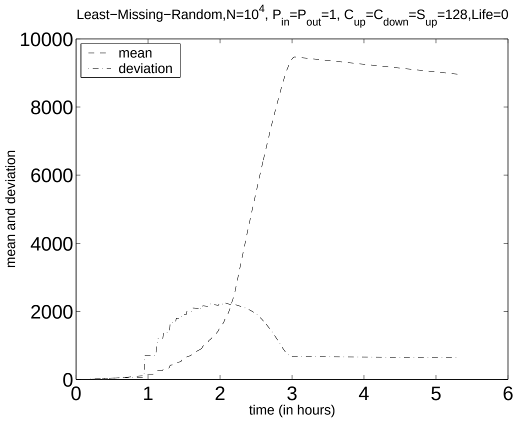


Figure 5.35: The number of completed clients against time. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

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 5.36(a). From figure 5.36(a) we can observe that the number of chunks



(a) Number of chunks vs time and chunk index



(b) Mean and deviation vs time

Figure 5.36: 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* $= 0$).

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 5.36(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 5.37). This figure is after 5.5

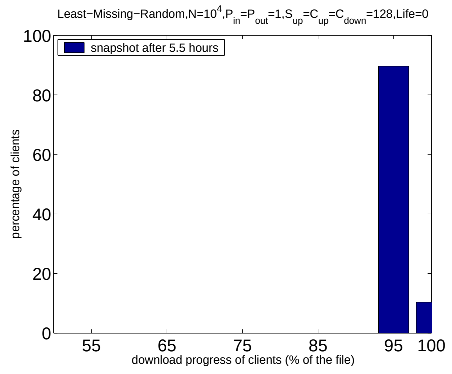


Figure 5.37: The download progress of clients against time for *Least Missing Random*. All N clients arrive at time $t = 0$. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download (*Life* $= 0$).

hours (9800 seconds) of simulations.

Conclusions

As we stated at the beginning of this chapter, 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.

5.6.6 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.

5.7 Simulation Results: Poisson Arrival of Clients

5.7.1 Basic Results

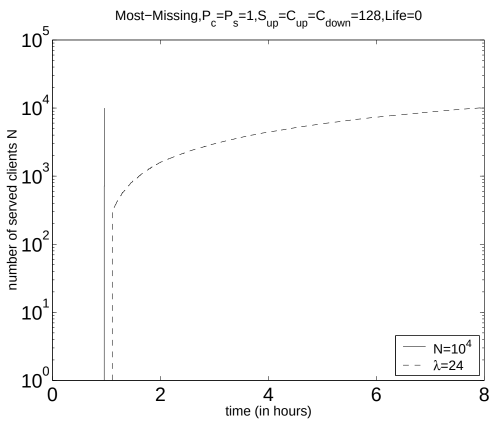
We now extend our analysis and study the scenario of continuous arrival of clients. We assume that clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. The parameter values that we consider in this basic and homogeneous scenario are presented in table 5.10.

Table 5.10: Parameter values.

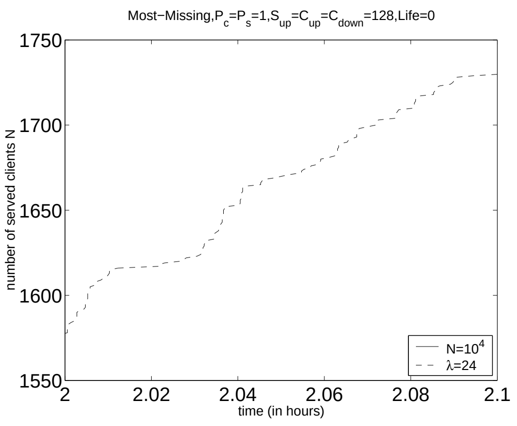
Arrivals Process	C_{up}	C_{down}	S_{up}	P_{in}	P_{out}	*Life*
$\lambda = 24$ clients/min	128 Kbps	128 Kbps	128 Kbps	1	1	0

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, 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 5.38(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 5.10, 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



(a) Overall view



(b) Zoom over a sub-set of clients

Figure 5.38: 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 ($Life = 0$).

have completed their download. From figure 5.38 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 5.38(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 5.39 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

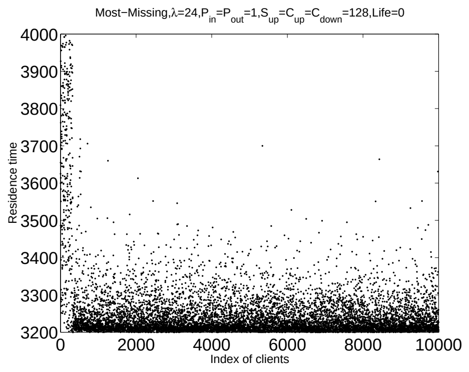
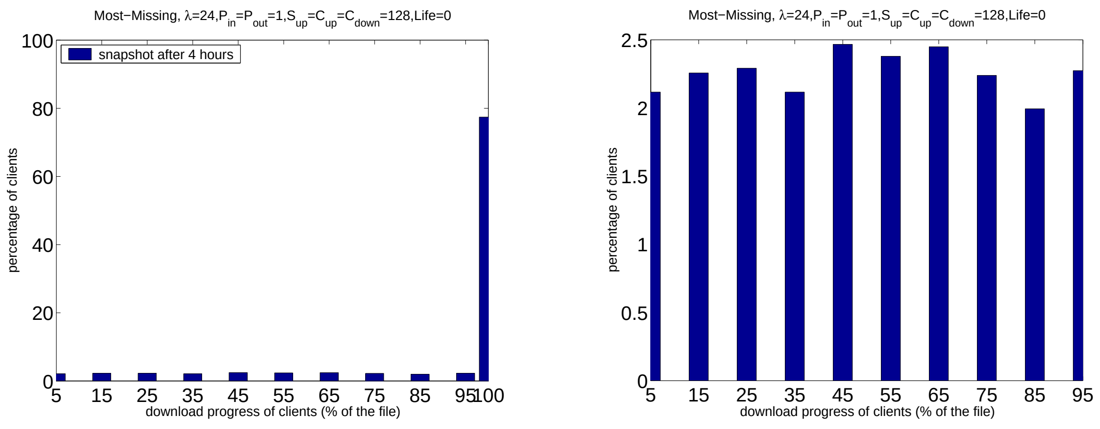


Figure 5.39: The residence time of clients for *Most Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file ($Life = 0$).

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,

[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.

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 5.40 where we give a snapshot of the download progress of clients after 4 hours of simulations. This figure shows how clients in progress are

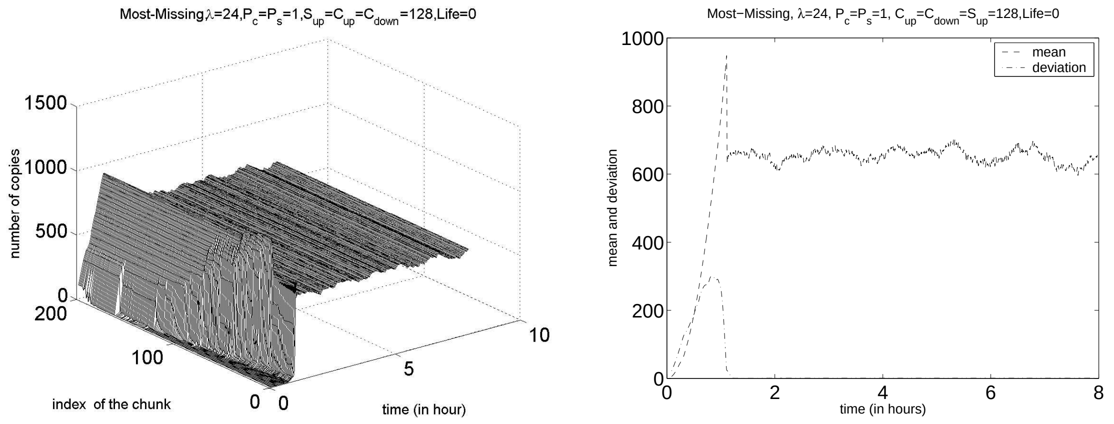


(a) overall view (b) zoom over non completed clients

Figure 5.40: 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 ($Life = 0$).

distributed over different batches, with each batch comprises similar number of clients.

As concerns the chunks evolution over time, figure 5.41(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



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

Figure 5.41: 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 ($Life = 0$).

new behavior here: Chunks that are injected later catch faster with earlier ones and consequently, the deviation drops faster to zero (figure 5.41(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 5.42). Within 8 hours of simulation, *Least Missing* serves **10163 clients** and *Most Missing* serves **10153 clients**. Under the parameter values of table 5.10, a perfect *Least Missing* (i.e. a single linear chain) serves only

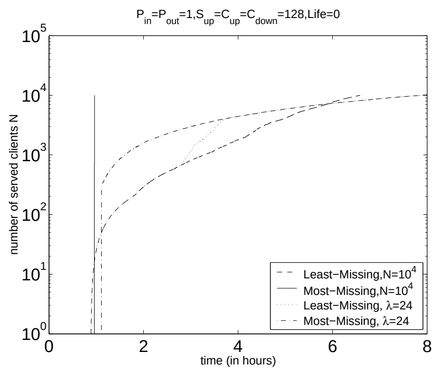


Figure 5.42: The number of completed clients against time. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

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 5.42 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 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 5.43).

The improvement in the performance of *Least Missing* can also be seen if we observe the chunk distribution over time in figure 5.44. As compared to an instantaneous arrival of clients (figure 5.14), figure 5.44 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.

[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.

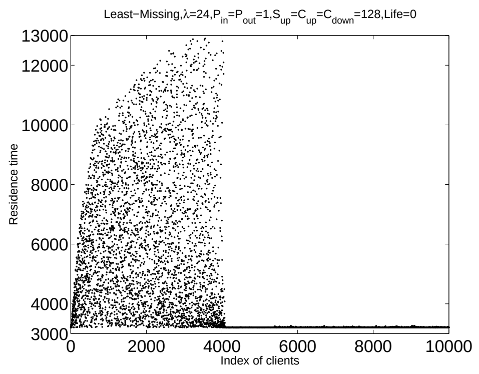
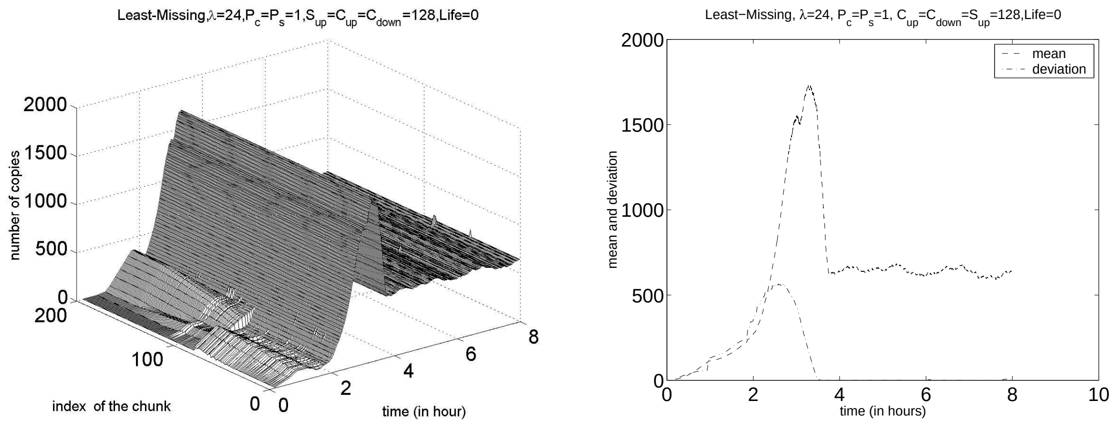


Figure 5.43: The residence time of clients for *Least Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ 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

Figure 5.44: 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).

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.

5.7.2 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 5.11) have exhibited the same tendency (figure 5.45).

Table 5.11: 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

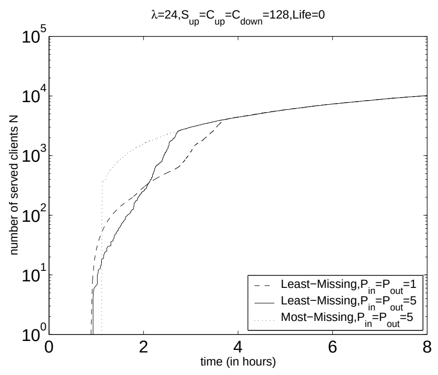


Figure 5.45: The number of completed clients against time. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

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 5.6.2 we know that large indegrees/outdegrees allow *Least Missing* to engage clients fast into the file delivery, which improves the system performance. But from section 5.7.1 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 5.46 summarizes what we just explained. The residence time of clients within the first few hours decreased

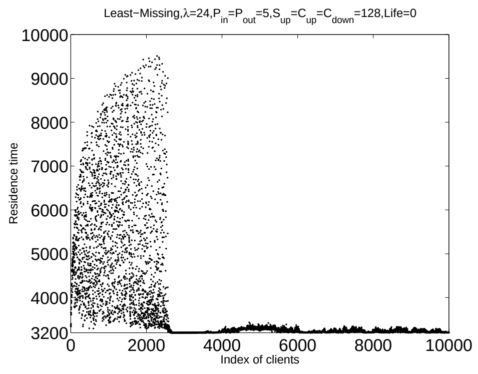


Figure 5.46: The residence time of clients for *Least Missing*. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most 5 download and 5 upload connections ($P_{in} = P_{out} = 5$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they receive the file (*Life* $= 0$).

while it did change for subsequent clients.

5.7.3 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 5.47 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 5.12.

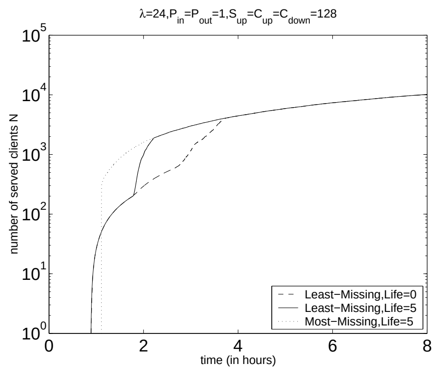


Figure 5.47: The number of completed clients against time. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps.

Table 5.12: 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 5.38(a), figure 5.47 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 5.47 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.

5.7.4 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 5.48 demonstrates that *Least Missing* performs very badly in heterogeneous environments. Table 5.13

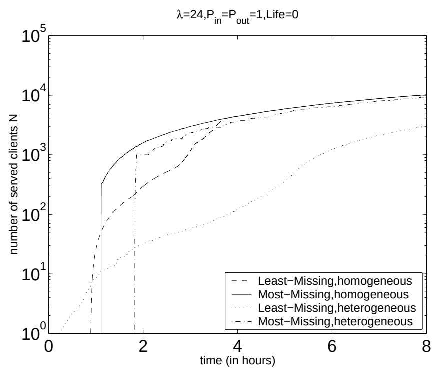


Figure 5.48: The number of completed clients against time. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). Clients disconnect as soon as they complete their download ($Life = 0$).

points out the values that we consider for the different parameters in this section.

Table 5.13: 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 5.48 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.

5.7.5 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 5.14. The scaling behavior of the two archi-

Table 5.14: 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

tectures (figure 5.49) confirm once again the importance of the chunk selection strategy as a key design for the system.

5.7.6 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*

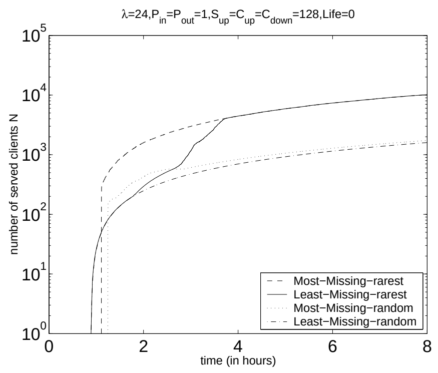


Figure 5.49: The number of completed clients against time. Clients arrive to the system according to a Poisson process with rate $\lambda = 24$ clients/min. Each peer maintains at most one download and one upload connection ($P_{in} = P_{out} = 1$). We assume homogeneous upload and download capacities $S_{up} = C_{up} = C_{down} = 128$ Kbps. Clients disconnect as soon as they complete their download ($Life = 0$).

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.

5.8 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 chapter 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 [36]: 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* [84], where each client reports to the server the information about the connection it is using.

5.9 Conclusions and Future Work

5.9.1 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 chapter 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 chapter, 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 chapter 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 chapter 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.

5.9.2 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* [26], 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* [50], 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.

Chapter 6

Summary and Conclusions

The huge increase in the number of clients using the Internet calls for efficient and scalable algorithms to distribute the content. Approaches for Scalable Content Distribution in the Internet was the focus of this thesis. We focused on two important services, VoD and file replication that have high resources requirements and induce high traffic.

First, we studied how to provide a large scale VoD service in dedicated overlay networks. Our contribution there was a new video distribution architecture where each video is split into two parts, the prefix and the suffix. The prefix is stored at the prefix servers and delivered to clients via a closed-loop scheme while the suffix is stored at the central suffix server and broadcast to clients using an open-loop scheme. The closed-loop algorithm that we use is *Controlled Multicast* and the open-loop algorithm is the *Tailored Periodic Broadcast*.

Given this architecture, we developed an analytical cost model, called PS-model, that permits to minimize the delivery cost of the video. The delivery cost of a video is the sum over the network bandwidth cost and the server cost consumed for storing and scheduling the video at the different servers. Note that in our cost model, we assume that the distribution network is organized as a tree with an outdegree m and l levels.

Using this cost model, we evaluated the performance of our PS architecture under various assumptions on the system parameters. Our results showed how the PS-model always finds the trade-offs between the cost for transmission and the cost for storage.

After having presented the basic scaling performance of our architecture, we used the PS-model to investigate many interesting extensions. The first extension is the scenario where the prefix servers can be placed at only fixed levels in the tree distribution. We derived the increase in the system cost under this scenario. Our conclusion is that the system performance can drop by up to 45%.

Another extension is the dimensioning scenario of a VoD system from scratch. We assumed an initial set $\mathcal{V} = \{1, \dots, K\}$ of K videos and computed the system resources in terms of storage and I/O at each level in the network. We then studied how the PS-architecture adapts to the change in the popularity distribution or in the number of videos K. For instance, in worst cases and when the number of videos goes from 100 to 150, the overall growth in the system remains low, below 30%.

Finally, we proved that PS-model can be easily extended to study different architectural choices like S-sat and P-hybrid. As compared to the PS architecture, the S-sat architecture transmits the suffix via satellite instead of a terrestrial network while the prefix is always stored at the prefix servers and distributed to clients through the Internet via *Controlled Multicast*. The benefits that S-sat can bring to the system are very important; the system cost has been dramatically discounted, by up to 80%. The second architectural choice was the P-hybrid architecture that takes advantage of the storage capacity available at the clients. We allowed clients to store locally, in their set-top boxes, the prefix part of some or all popular videos in the system. The reduction in the system cost achieved by P-hybrid was in the order of $30-45\%$.

We then continued our study of issues related to VoD services but in a P2P context (rather than dedicated overlay networks). Most of existing work in this area advised to construct a multicast tree to deliver the video to clients. Such solutions face the challenge of constructing and maintaining the multicast tree in an environment where clients can leave the system at any time. Our contribution there was a new pull-based approach, referred to as PBA, that makes the server as simple as possible and, at the same time, achieves a very good performance.

The basic idea of PBA is quite simple. When a new client wishes to receive a video, it first contacts the server. If there is enough free bandwidth along the path to the new client, the server transmits the video to the new client. Otherwise, the server responds to the new client with a list of C_s servants. These servants are clients that have received or are currently receiving the video. Then, the new client searches for an available servant from which it downloads the video.

We evaluated the efficiency of PBA and compared it to P^2Cast, a multicast tree-based approach. Our results showed that PBA not only simplifies the server, but it also consumes low network bandwidth resources and allows a lot of clients to access the service. We also investigated the gain in the performance of PBA when each client receives the video from a servant that is close in terms of physical distance. This extension, referred to as CSF, reduces significantly the rejection probability and the system cost.

The last contribution of this thesis was an analysis of the use of P2P networks for file replication services. Existing solutions can be broadly classified into tree-based and mesh-based approaches. The first part of our analysis was a comparison between the performance of both kinds of approaches. For this purpose, we introduced two mesh-based architectures, namely *Least Missing* and *Most Missing*. The *Least Missing* architecture serves in priority clients that have the largest number of chunks while *Most Missing* gives more priority to new comers.
We analytically demonstrated that the number of served clients with *Least Missing* can scale as in a tree with an outdegree k. The key idea is to set the indegree of clients to $P_{in} = 1$ and the outdegree to $P_{out} = k$. We also provided a comparison between *Most Missing* and $PTree^k$, the most efficient tree architecture as proved in [18]. Our results clearly revealed that *Most Missing* behaves similarly as $PTree^k$ while avoiding the construction of parallel trees.

The second part of our analysis was a set of simulation results where we evaluated the two architectures *Least Missing* and *Most Missing* under different scenarios. The results that we presented showed clearly the importance of the cooperation strategy between clients and the chunk selection process on the behavior of the system. We also isolated the main parameters that influence the performance of mesh approaches namely, the arrival process of clients, their indegree/outdegree, their life time, and their bandwidth heterogeneity. Our study provides new insights that help in the conception of new cooperative architectures depending on the goals and the environment conditions.

Bibliography

[1] "About Akamai", http://www.akamai.com/en/html/about/overview.html.

[2] "Gnutella", http://www.gnutella.com/.

[3] "KaZaA", http://www.kazaa.com/.

[4] "Napster", http://www.napster.com/.

[5] "FreeFlow: How it Works", Akamai, Cambridge, MA, USA, November 1999.

[6] "What is TiVo: Technical Aspects", http://www.tivo.com/, 2003.

[7] "BitTorrent beats Kazaa, accounts for 53% of P2P traffic", AlwaysOn Site, July 2004, http://www.alwayson-network.com/index.php.

[8] E. Adar and B. A. Huberman, "Free Riding on Gnutella", *First Monday*, 5(10), October 2000.

[9] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On Optimal Batching Policies for Video-on-Demand Storage Servers", In *Proc. of ICMCS*, pp. 253–258, Hiroshima, Japan, June 1996.

[10] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems", In *Proc. of ICMCS*, pp. 118–126, Hiroshima, Japan, June 1996.

[11] J. M. Almeida, D. L. Eager, M. C. Ferris, and M. K. Vernon, "Provisioning Content Distribution Networks for Streaming Media", In *Proc. of INFOCOM*, NY, USA, June 2002.

[12] K. C. Almeroth and M. H. Ammar, "Providing a Scalable Interactive VOD Service Using Multicast Communication", In *Proc. of IC3N*, pp. 292–301, San Francisco, CA, USA, September 1994.

[13] D. P. Anderson, "Metascheduling for continuous media", *In Proc. of ACM/Transaction on Computer Systems*, 11(3):226–252, August 1993.

[14] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast", In *Proc. of ACM SIGCOMM*, Pittsburch, PA, USA, August 2002.

[15] S. Banerjee, J. Brassil, A. C. Dalal, S. J. Lee, E. Perry, P. Sharma, and A. Thomas, "Rich Media from the Masses", HPL-2002-63R1, HP Lab, Palo Alto, CA, USA, May 2002.

[16] S. A. Barnett and G. J. Anido, "A cost comparison of distributed and centralised approaches to video on demand", *In Proc. of IEEE Journal on Selected Areas in Communications*, 14(6):1173–1183, August 1996.

[17] E. W. Biersack, A. Jean-Marie, and P. Nain, "Open-loop Video Distribution with Support of VCR Functionnality", *In Proc. of Performance Evaluation*, 49(1-4):411–428, September 2002.

[18] 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.

[19] Y. Birk and R. Mondri, "Tailored Transmissions for efficient Near-Video-on-Demand Service", In *Proc. of ICMCS*, pp. 226–231, Florence, Italy, June 1999.

[20] Y. Cai, K. Hua, and K. Vu, "Optimizing Patching Performance", In *Proc. of MMCN*, pp. 204–216, San Jose, California, USA, January 1999.

[21] 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.

[22] G. Chan and F. Tobagi, "Distributed Servers Architectures for Networks Video Services", *In Proc. of IEEE/ACM Transactions on Networking*, 9(2):125–136, April 2001.

[23] L. Cherkasova and J. Lee, "FastReplica: Efficient Large File Distribution within Content Delivery Networks", In *Proc. of USITS*, seattle, WA, USA, March 2003.

[24] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture", In *Proc. of ACM SIGCOMM*, San Diago, CA, USA, August 2001.

[25] Y. H. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast", In *Proc. of ACM SIGMETRICS*, pp. 1–12, Santa Clara, CA, USA, June 2000.

[26] 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.

[27] P. Cudre-Mauroux and K. Aberer, "A Decentralized Architecture for Adaptive Media Dissemination", In *Proc. of ICME*, Lausanne, Switzerland, August 2002.

[28] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching", In *Proc. of ACM Multimedia*, pp. 15–23, San Francisco, CA, USA, October 1994.

[29] Dash Optimization, *Xpress-Mp Essentials*, February 2002.

[30] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming Live Media over a Peer-to-Peer Network", 31, Stanford University, USA, 2001, http://dbpubs.stanford.edu:8090/pub/2001-31.

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

[32] D. L. Eager, M. Ferrris, and M. K. Vernon, "Optimized Regional Caching for On-Demand Data Delivery", In *Proc. of MMCN*, pp. 301–316, San Jose, CA, USA, January 1999.

[33] D. L. Eager and M. K. Vernon, "Dynamic Skyscraper Broadcasts for Video-on-demand", In *proc. of MIS*, volume 1508 of *LNCS*, pp. 18–32, Istanbul, Turky, September 1998, Springer Verlag.

[34] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Minimizing Bandwidth Requirements for On-Demand Data Delivery", In *Proc. of MIS*, Indian Wells, CA, USA, October 1999.

[35] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Optimal and Efficient Merging Schedules for Video-on-Demand Servers", In *Proc. of ACM Multimedia*, pp. 199–202, Orlando, FL, USA, November 1999.

[36] P. A. Felber and E. W. Biersack, “Self-scaling Networks for Content Distribution”, In *Proc. of Self-**, Bertinoro, Italy, May 2004.

[37] L. Gao and D. Towsley, “Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast”, In *Proc. of ICMCS*, pp. 117–121, Firenze, Italy, June 1999.

[38] L. Garcés-Erice, K. W. Ross, E. W. Biersack, P. A. Felber, and G. Urvoy-Keller, “Topology-Centric Look-Up Service”, In *Proc. of COST264/ACM NGC*, pp. 58–69, Munich, Germany, September 2003.

[39] A. Gelman and S. Halfin, “Analysis of Resource Sharing in Information Providing Services”, In *Proc. of Globecom*, pp. 312–316, San Diego, CA, USA, December 1990.

[40] 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.

[41] L. Golubchik, J. C. S. Lui, and R. Muntz, “Reducing I/O Demand in Video-on-Demand Storage Servers”, In *Proc. of ACM SIGMETRICS*, pp. 25–36, Ottawa, Canada, May 1995.

[42] V. K. Goyal, “Multiple description coding: compression meets the network”, *In Proc. of IEEE Signal Processing Magazine*, 18(5):74–93, September 2001.

[43] K. P. Gummadi, S. Saroiu, and S. D. Gribble, “King: Estimating Latency between Arbitrary Internet End Hosts”, In *Proc. of SIGCOMM IMW*, Marseille, France, November 2002.

[44] Y. Guo, S. Sen, and D. Towsley, “Prefix Caching assisted Periodic Broadcast: Framework and Techniques for Streaming Popular Videos”, In *Proc. of ICC*, New York, USA, April 2002.

[45] Y. Guo, K. Suh, J. Kurose, and D. Towsley, “P2Cast: Peer-to-Peer Patching Scheme for VoD Service”, In *Proc. of WWW*, Budapest, Hungary, May 2003.

[46] B. Hill, “P2P: 70-80 Percent of All EuroNet Traffic”, The Digital Music Weblog, May 2004, http://digitalmusic.weblogsinc.com/.

[47] A. Hu, “Video-on-Demand broadcasting protocols: A comprehensive study”, In *Proc. of INFOCOM*, volume 1, pp. 508–517, Anchorage, Alaska, USA, April 2001.

[48] K. A. Hua, Y. Cai, and S. Sheu, “Patching : A Multicast Technique for True Video-on-Demand Services”, In *Proc. of ACM Multimedia*, pp. 191–200, Bristol, UK, September 1998.

[49] K. A. Hua and S. Sheu, “Skyscraper Broadcasting for Metropolitan VOD”, In *Proc. of ACM SIGCOMM*, August 1997.

[50] 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.

[51] J. Jannotti, D. K. Gifford, and K. L. Johnson, “Overcast: Reliable Multicasting with an Overlay Network”, In *Proc. of OSDI*, San Diago, CA, USA, October 2000.

[52] L. Juhn and L. Tseng, “Fast Data Broadcasting and Receiving Scheme for Popular Video Service”, *In Proc. of IEEE Transactions on Broadcasting*, 44(1):100–105, March 1998.

[53] L. S. Juhn and L. M. Tseng, “Harmonic Broadcasting for Video-on-Demand Service”, *In Proc. of IEEE Transactions on Broadcasting*, 43(3), September 1997.

[54] T. Kameda and R. Sun, "Survey on VoD Broadcasting Schemes", 2003, http://www.cs.sfu.ca/CC/886/tiko/lecnotes/survey.pdf.

[55] K. Kong and D. Ghosal, "Mitigating Server-Side Congestion in the Internet through Pseudoserving", *In Proc. of IEEE/ACM Transactions on Networking*, pp. 530–544, August 1999.

[56] B. Krisnamurthy and J. Wang, "On Network-Aware Clustering of Web Sites", In *Proc. of ACM SIGCOMM*, Stockholm, Sweden, August 2000.

[57] M. Krunz and S. K. Tripathi, "Bandwidth allocation strategies for transporting variable-bit-rate video traffic", *In Proc. of IEEE Communications Magazine*, 37(1):40 – 46, January 1999.

[58] J. Lee and G. de Veciana, "Adaptive Fast Content Replication within Content Delivery Networks", April 2004.

[59] F. Li and I. Nikolaidis, "Trace-adaptive fragmentation for periodic broadcast of VBR video", In *Proc. of NOSSDAV*, Basking Ridge, New Jersey, USA, June 1999.

[60] J. Liang, R. Kumar, and K. W. Ross, "Understanding KaZaA", 2004, Polytechnic University Brooklyn, NY, USA, Submitted for publication.

[61] T. S. E. Ng and H. Zhang, "Towards global network positioning", In *Proc. of ACM SIGCOMM Workshop on Internet Measurement*, pp. 25–29, San Francisco, CA, USA, August 2001, ACM Press.

[62] A. Nicolosi and S. Annapureddy, "P2PCast: A Peer-to-Peer Multicast Scheme for Streaming Data", In *Proc. of IRIS Student Workshop*, Cambridge, MA, UK, August 2003.

[63] J. Nonnenmacher, "Personal communication", October 2002.

[64] J. Nussbaumer, B. Patel, F. Schaffa, and J. Sterbenz, "Networking Requirements for Interactive Video on Demand", *In Proc. of EEE Journal on Selected Areas in Communications*, 13(5):779–787, June 1995.

[65] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for internet hosts", In *Proceedings of ACM SIGCOMM*, pp. 173–185, San Diego, CA, USA, August 2001, ACM Press.

[66] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking", In *Proc. of NOSSDAV*, Miami Beach, FL, USA, May 2002.

[67] J.-F. Pâris, "A Broadcasting Protocol for Compressed Video", In *Proc. of Euromedia*, pp. 78–84, Munich, Germany, April 1999.

[68] J. F. Pâris, S. W. Carter, and D. D. E. Long, "A Hybrid Broadcasting Protocol for Video on Demand", In *Proc. of MMCN*, pp. 317–326, San Jose, CA, USA, January 1999.

[69] J. Pâris, "A Simple Low-Bandwidth Broadcasting Protocol for Video-on-Demand", In *Proc. of IC3N*, pp. 118–123, Boston-Natick, MA, USA, October 1999.

[70] J. Pâris, S. W. Carter, and D. D. E. Long, "Efficient Broadcasting Protocols for Video on Demand", In *Proc. of MASCOTS*, pp. 127–132, Montreal, Canada, July 1998.

[71] J. Pâris, S. W. Carter, and D. D. E. Long, "A Low Bandwidth Broadcasting Protocol for Video on Demand", In *Proc. of IC3N*, pp. 690–697, Lafayette, LA, USA, October 1998.

[72] J. Pâris, D. D. E. Long, and P. E. Mantey, "Zero-Delay Broadcasting Protocols for Video-on-Demand", In *Proc. of ACM Multimedia*, pp. 189–197, Orlando, FL, USA, November 1999.

[73] R. Rajendran and D. Rubenstein, "Optimizing the Quality of Sclable Video Streams on P2P Networks", In *Proc. of Globecom*, Dallas, TX, USA, November 2004.

[74] S. Ramesh, I. Rhee, and K. Guo, "Multicast with Cache (MCache): An Adaptive Zero Delay Video-on-Demand Service", *In Proc. of IEEE Transactions of Circuit and System of Video Transmission*, 11(3), March 2001.

[75] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", In *Proc. of ACM SIGCOMM*, San Diago, CA, USA, August 2001.

[76] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Application-Level Multicast Using Content-Addressable Networks", In *Proc. of NGC*, pp. 14–29, London, UK, November 2001.

[77] R. Rejaie and A. Ortega, "PALS: Peer to Peer Adaptive Layered Streaming", In *Proc. of NOSSDAV*, Monterey, CA, USA, June 2003.

[78] P. Rodriguez, A. Kirpal, and E. W. Biersack, "Parallel-Access for Mirror Sites in the Internet", In *Proc. of INFOCOM*, Tel-Aviv, Israel, March 2000.

[79] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", In *Proc. of Middleware*, pp. 329–350, Heidelberg, Germany, November 2001.

[80] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The design of a large scale event notification infrastructure", In *Proc. of NGC*, UCL, London, November 2001.

[81] D. Saparilla, K. W. Ross, , and M. Reisslein, "Periodic Broadcasting with VBR-Encoded Video", In *Proc. of INFOCOM*, pp. 464–471, New York, USA, March 1999.

[82] L. E. Schrage, "A Proof of the Optimality of the Shortest Remaining Service Time Discipline", *Operations Research*, 16:670–690, 1968.

[83] N. Shankar, C. Komareddy, and B. Bhattacharjee, "Finding Close Friends over the Internet", In *Proc. of ICNP*, Mission Inn, Riverside, CA, USA, November 2001.

[84] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol", In *Proc. of INFOCOM*, Hong-Kong, China, March 2004.

[85] S. Sheu, K. A. Hua, and W. Tavanapong, "Chaining: A Generalized Batching Technique for Video-On-Demand Systems", In *Proc. of ICMCS*, pp. 110–117, Ottawa, Canada, June 1997.

[86] D. Sitaram and A. Dan, *Multimedia Servers: Applications, Environments, and Design*, Published by Morgan Kaufmann, October 1999.

[87] A. Stavrou, D. Rubenstein, and S. Sahu, "A Lightweight, Robust P2P System to Handle Flash Crowds", In *Proc. of ICNP*, Paris, France, November 2002.

[88] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications", In *Proc. of ACM SIGCOMM*, San Deigo, CA, USA, August 2001.

[89] D. A. Tran, K. A. Hua, and T. T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming", In *Proc. of INFOCOM*, San Francisco, CA, USA, March 2003.

[90] S. Viswanathan and T. Imielinski, "Pyramid Broadcasting for Video On Demand Service", In *Proc. of MMCN*, San Jose, CA, USA, February 1995.

[91] B. Wang, S. Sen, M. Adler, and D. Towsley, "Proxy-based Distribution of Streaming Video over Unicast/Multicast Conncetions", 01-05, UMass CMPSCI, March 2001.

[92] P. P. White and J. Crowcroft, "Optimized Batch Patching with Classes of Service", *In Proc. of ACM SIGCOMM Computer Communication Review*, 30(5), October 2000.

[93] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming", In *ICDCS*, pp. 363–371, Washington - Brussels - Tokyo, July 2002.

[94] X. Yang and G. de Veciana, "Service Capacity of Peer-to-Peer Networks", In *Proc. of INFOCOM*, Hong-Kong, China, March 2004.

[95] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork", In *Proc. of INFOCOM*, pp. 594–602, Boston, MA, USA, March 1996.

[96] Y. Zhao, D. L. Eager, and M. K. Vernon, "Efficient Delivery Techniques for Variable Bit Rate Multimedia", In *Proc. of MMCN*, San Jose, CA, USA, January 2002.

[97] Y. Zhao, D. L. Eager, and M. K. Vernon, "Network Bandwidth Requirements for Scalable On-Demand Streaming", In *Proc. of INFOCOM*, New York, USA, june 2002.