U.F.R Informatique
Ecole Doctorale: Sciences Mathématiques de Paris Centre
UNIVERSITE PARIS DIDEROT (PARIS 7)

DOCTORAT EN INFORMATIQUE

par **Diego Perino**

On resource allocation algorithms for peer-to-peer multimedia streaming

Algorithmes d'allocation de ressources pour le streaming en pair-à-pair

Soutenue publiquement le 16 Novembre 2009 devant la commission d'examen composée de

Rapporteurs :	Marco Ajmone Marsan	*Professeur, Politecnico di Torino*
	Pascal Felber	*Professeur, Université de Neuchâtel*
	Anne-Marie Kermarrec	*Directrice de recherche, INRIA*
Examinateurs :	Pierre Fraignaud	*Directeur de recherche, CNRS*
	Arnaud Legout	*Chargé de recherche, INRIA*
	Laurent Massoulié	*Chercheur, Thomson Lab*
Directeurs :	Fabien Mathieu	*Chercheur, Orange Labs*
	Laurent Viennot	*Directeur de recherche, INRIA*

Acknowledgements

This is the first time I have the opportunity, and especially the time, to write an acknowledgement section to thank people for their contribution to the accomplishment of my work. I hope I won't forget any, but if your name does not appear below and you feel you have contributed in some way to my work, I thank you as well.

Thanks to Alice, that has been at my side during all the period of my Ph.D. She shared with me the best moments, as well as the hardest ones, always tolerating my moods. She constantly encouraged me giving me the strength and the stability needed to complete this thesis.

Thanks to my parents, Giovanna and Piergiorgio, and to my family that gave me the opportunity to pursue my dreams even if they drove me being far from them. No matters where I will be, I know you will always support me and I will continuously be with you.

Thanks to all my friends! A special thank to Valerio and Matteo, that shared with me lot of my Parisian life, to Matteo, not only my flat-mate in Paris but a friend during all our long path since we left Italy, and to Dario, Roberto, Silvia and Giovanna, my childhood friends that always encouraged me in my choices and that I am very pleased to meet during all my trips to Susa, my home town. A special thank also to the anonymous organizers of the "Erasmus Party @ Mix" that allowed me to relax "Every Thursday" during these Ph.D. years.

Thanks to Fabien, my "industrial" advisor: he taught me lot of things, from technical subjects to solutions of everyday life issues in France. In particular, he taught me to work with excitement and pleasure on research topics, being motivated everyday to face new challenges in order to obtain the best possible outcomes. Without this "taste for research" I would not be able to obtain the results reported in this manuscript. Thanks to Laurent, my "academic" advisor. He has always be available for me and has given me precious advices and recommendations to improve my work.

Thanks to Fabio that has been my industrial advisor during my master thesis. The work carried on with him has probably been decisive to convince me to undertake a Ph.D. Thanks also to Joaquin and Gwendal who were in my team during that period and that also played an important role on this choice. And thanks to Ernst, my academic advisor during my master thesis, for his precious recommendations on my master and Ph.D. work.

Thanks to the TRM team at Orange Labs: this group of people provides me a very challenging and exiting environment to develop my Ph.D. work. It has been possible to discuss and exchange ideas with very valuable people as well as spending good time with them playing soccer or during "barbeques" and "pots". I hope I will found such an environment in my future working experiences. A special thank to Luca: it was a pleasure working with him on a topic that is unfortunately not reported in this thesis for lack of space and about which he taught me a lot. I really liked our "italian" discussions about different subjects related to research and not.

Thanks to the GANG team, I really enjoyed working with them. Our meetings gave me different point of views and technical approaches to face research problems, coming from people with a different background than mine. I really liked the fact that our meetings normally started or ended with a "déjeuner" or "apéro" at "la ficelle".

Thanks to all my co-authors! It has been a pleasure and an honor to work with them. I have learnt many things during our collaborations and I hope I will have the opportunity to work with them again in future.

And finally, I would like to thank all the "jury" members to have accepted this duty, especially the "rapporteurs" that have read such a long and dense manuscript.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

During the late 90's the growing popularity of Internet, the increase of access bandwidths and the development of new technologies have made **streaming media** possible and affordable for ordinary customers. *Streaming a media content* consists in transmitting a continuous flow of multimedia data that users can play out *while they are retrieved* without waiting for the entire content to be downloaded.

The 2000's have seen an increasing popularity of *multimedia streaming*: from first websites offering the stream of a short media file, to the advent of live streaming applications, User Generated Content (UGC) services and mobile TV. Streaming traffic represents today the largest part of Internet traffic and is supposed to increase even more thanks to large deployment of High Definition content (HD), of high-speed access and home wireless technologies, like FTTH, IEEE802.11n and WirelessHD.

In this thesis we consider streaming applications based on a peer-to-peer architecture, which can provide the increasing amount of resource required to realize a streaming service, and effectively deal with the growing number of users.

1.1 Multimedia streaming

The earliest technique employed to transfer a media content was simply **bulk media distribution**, a mechanism where contents are considered as common files without any special coding or ordering requirements. Media files are retrieved with traditional techniques like FTP or HTTP, or downloaded by means of file-sharing applications like BitTorrent, E-donkey, or Gnutella, without any specific time or rate constraints.

On the contrary, in **multimedia streaming** the media content is coded as a continuous flow of data and is characterized by a *stream bitrate*: this is the data rate at which the content is encoded and indicates the amount of data per second required for play out. Note that the stream bitrate is not necessarily constant and may vary over time. Data should be downloaded sequentially and users may play out the content while it is retrieved without waiting for the end of download.

Media streaming may be broadly classified into three categories: *on-demand*, *live* and *interactive*.

On-demand streaming users are interested in media contents that are fully available somewhere. The media files are usually stored in a *catalog* of contents, and users can retrieve any content from the catalog at anytime. A first challenge is to *store* this catalog so that any file is always accessible by every customer. Moreover, the size of the catalog provided should be as large as possible in order to attract a wide range of customers. An on-demand streaming system should *minimize the start-up delay*, which is the time between the moment at which a user selects a given content and the moment the content playback begins. Once the playback is started, the system should *guarantee its continuity* so that users do not experience blocking events i.e. frozen frames. To this purpose the download rate of every content should be at least equal to its stream rate in order to retrieve enough data on time to correctly decode the stream[1]. An on-demand system may also provide *video seeking functionalities* allowing users to jump from one point of the content to another.

In **live streaming** a multimedia content is not available in advance because it is generated by a source on the fly while a live event is going on. It should be distributed to users as fast as possible because data is interesting and useful only for a limited period of time. This introduces an additional time requirement: a live streaming system should *minimize the playout delay* which is the time between the moment at which the content is generated by the source and the moment the content is played out at users. Contrary to on-demand streaming, content should not be stored and *storage constraints are relaxed*; nevertheless *start-up delay and playback continuity* are also very important in the live streaming context. To this purpose the rate at which the content is retrieved should be equal to its stream bitrate; however, unlike on-demand streaming, the effective download rate cannot be larger than this rate, because data is not yet available and cannot be retrieved in advance. Moreover, in a live streaming system users are interested in the same data simultaneously; this synchronization may help the distribution process thanks to possible data exchanges between the different clients.

The **interactive streaming** is the most time sensitive streaming application. The requirements are very similar to the live case, but the distribution latency here cannot exceed 100-150 milliseconds [22]. In fact, heavy user interaction demands fast responses between actions and reactions. Conferencing applications are typical examples of interactive streaming.

It is clear that all media streaming applications should deal with **strict time requirements** and guarantee the **smothness of playback**. The data of a multimedia content should arrive to users at a sufficient rate to allow its continuous playback, and the system should provide them as fast as they have been generated or shortly after the user request. The challenge for a streaming system is *to transfer* the data from the locations where it is available to the clients, under the constraints mentioned above.

As in most network applications the factors limiting data transfers are the *network bandwidth* of the system and *the way this bandwidth is exploited*. In on-demand streaming the *storage capacity* is an additional constraint because media contents are available in advance, and *storage mechanisms* also play an important role on performance. The performance of a streaming application is thus largely determined by the management of critical resources such as **network and storage capacities**.

[1]With special coding techniques, a download rate lower that the nominal stream rate allows to decode the stream anyway (e.g. FEC coding like [99]), or different download rates lead to different qualities of the media content (e.g. layered or multiple-description coding [20]).

1.2 Delivery of multimedia contents

Internet media traffic has been steadily growing in the past few years. It is reported to currently double every 3-4 months [2], and expected to increase tenfold from 2008 to 2013 [36]. Unlike Web traffic, multimedia can be transferred by means of several architectures and approaches, like multicast overlays, peer-to-peer applications, or CDNs. These solutions may overcome the limitations imposed by a centralized client-server approach, where storage and bandwidth resources may be insufficient to deal with large audiences.

A special diffusion technique is IP multicast which is the best approach for a one-to-may communication over an IP infrastructure. A delivery tree rooted at the source towards the clients is built directly in the network and optimizes the network utilization. This solution is used today for the distribution of IPTV to the customers of a given ISP. However, large scale deployment of IP multicast has failed mainly because of lack of ISPs commercial interests and security issues [35].

IP multicast infrastructures are not always available, and other approaches must be employed. To select the most suitable one it is necessary to understand the main characteristics of media traffic. Guo *et al.* [46] perform an analysis over a very large set of media workloads and observe that *the rank of media objects follows a stretched exponential distribution*. This distribution has two parameters representing the aging of media accesses and the size of the media files. The authors deduce that as a consequence of this rank distribution the traditional caching of Web objects is not effective for multimedia contents, and there is a great potential to improve performance of client-side caching.

As a conclusion, authors state that *a performance-effective and cost-efficient media caching system should be capable of scaling its storage size as its workload increases over a long time period*. They therefore claim that **a peer-to-peer based streaming system can be very effective to deliver media contents**. In fact, bandwidth and storage capacities of a peer-to-peer system increase with the number of clients, thus being able to cope with media traffic patterns.

Considering the multimedia traffic evolutionary trends, and the potential offered by a peer-to-peer architecture, in this thesis we will consider live and on-demand streaming systems based on a peer-to-peer solution. In particular, we are going to deeply analyze the mesh-based approach for live streaming applications, and a "fully" peer-to-peer architecture for VoD, where users collaborate to store the video catalog and serve video requests generated by other users.

1.3 Peer-to-Peer networks

Peer-to-peer overlay networks are distributed systems running on top of Internet. They go beyond the traditional client-server concept: every node is in fact both client and server, sharing part of its resources to realize a given application. This approach may be used to alleviate servers' load, or to provide a given service without the existence of centralized entities. In the first scenario we say the application is *peer-assisted* or *hybrid*, while in the latter case we say the system is based on a *fully peer-to-peer* approach.

A P2P architecture provides massive scalability because its resources increase with the number of participants. Moreover, it may offer several features, like robust routing, efficient search,

selection of nearby peers, redundant storage, fault-tolerance, etc., that cannot be provided by a centralized system.

The price to pay is that the decentralized nature of P2P networks introduces new challenges on the management of system resources. First of all, *the maintenance of the overlay*: nodes join and leave at any time, and considering the potentially high number of participants they may have a limited view of the whole system. Peer discovery mechanisms and algorithms for overlay optimization are therefore required in order to provide self-organization, robustness and reliability.

On top of the overlay a set of algorithms is responsible for *task scheduling, content routing and data management*. Meta-data are used to describe the content stored across the peers and to represent local information. Finally, tools and applications are implemented with specific functionalities on top of this peer-to-peer substrate.

Resource allocation algorithms run at every node on the basis of local information and are designed to provide the aforementioned functionalities in order to set up the desired application.

There are two classes of peer-to-peer networks: *structured* and *unstructured*.

Structured peer-to-peer networks. In a structured P2P network the overlay topology and the relation between data and peers are strictly defined. These systems often use a Distributed Hash Table (DHT) as a substrate with different data structure. The basic component is a key space: a unique identifier (key) is assigned to each participant and to every data object. Every node is responsible to manage information and register the locations of objects whose keys are close to its ID. Every node has a small routing table representing a partial view of the system: the set of links contained on node routing tables forms the overlay network. Lookup and routing is performed traversing several nodes, approaching the destination key at every hop until the key is reached. This kind of routing is called *key-based* routing. Examples of structured peer-to-peer networks are Kademlia [77], Chord [106], and Splitstream [18].

Unstructured peer-to-peer networks. In an unstructured P2P network the links between the different peers are not strictly defined but established in an arbitrary manner. The network uses flooding-like routing, gossiping or epidemic-style mechanisms to forward data and control messages across the overlay. These mechanisms are very effective in coping with churn and unpredictable peer behaviors: churn can even be considered a good property of the system to optimize the overlay and improve stability.

Lookup techniques in unstructured networks are efficient to locate very popular contents but they may fail to locate rare objects; moreover the load on each node may grow linearly with the total number of queries and with the system size. Nevertheless, unstructured lookup systems are more commonly used than structured one because even if an unstructured approach may fail to locate rare objects, it generates much less overhead for popular contents. Moreover the use of *super-nodes* may mitigate the rare object issue.

Examples of unstructured peer-to-peer networks are BitTorrent [14], Gnutella [37], PPLive [95].

There are mainly two contexts in which a peer-to-peer architecture can be used: *controlled* and *uncontrolled* environments.

In **controlled environments** nodes have predictable behaviors and there is not need to provide incentives to collaborate. A central authority manages the whole system and there is a stable set of peers. An important example of this controlled scenario is the collection of residential gateways and set-top boxes installed by an ISP at user homes.

In **uncontrolled environments**, like a set of user PCs, the overlay does not arise from the collaboration of well known and connected groups of users, there is no central authority controlling the system, nor a reliable and stable set of resources shared by nodes. In fact, peers join and leave the system in unpredictable ways, and provide a variable and unpredictable amount of resources.

1.4 Thesis organization and contributions

The thesis is organized in two main parts. The first part is devoted to *mesh-based live streaming* systems, while the second part considers *on-demand streaming* applications. We do not consider in this thesis the interactive streaming. We believe this kind of streaming is extremely difficult to set up using a distributed approach because of hard time constraint, and the high sensitivity to packet losses and network congestion.

Before focusing on a specific streaming application, in Chapter 2 we analyze the performance that various peer-to-peer based systems can achieve from a bandwidth budget perspective. The *network bandwidth* is in fact a critical resource for both on-demand and live streaming systems. In peer-to-peer or peer-assisted applications the sum of data uploaded by nodes is equal to the sum of data they download: this is the *bandwidth conservation law*. Resource allocation algorithms are designed to exploit this bandwidth as much as possible, but they cannot overcome limitations imposed by the available capacity. We use the bandwidth conservation law as a starting point to derive explicit theoretical bounds on the achievable performance. The model we propose is quite general so that it can describe the performance of live and on-demand streaming systems, and file-sharing applications as well. We show that a service provider can increase the capacity of its network while reducing its costs by using a (hybrid) peer-to-peer architecture, and our results can serve as guidelines for the design and dimensioning of real systems.

1.4.1 Mesh-based peer-to-peer live streaming

The first part of the thesis is introduced by Chapter 3. We retrace the history of peer-to-peer live streaming from early systems, deployed around 2000, to increasing popularity of the recent commercial applications. We consider P2P live streaming in *uncontrolled environments*, where peers have unpredictable behaviors and share different amount of bandwidth. Distribution techniques should be **resilient** to this dynamic and unpredictable scenario in order to avoid playback disruption when network conditions change. These mechanisms should provide **incentive** to peers to cooperate in order to have enough bandwidth in the system to serve the media content to all nodes. To this purpose, peers contributing more resources should experience better media quality or shorter reception delays.

The **unstructured mesh-based** approach seems to be the most suitable to meet these requirements: it has been employed for the deployment of the most popular commercial streaming applications and has been shown efficient over a larger population of users with respect to a structured tree-based solution.

In Chapter 4 we investigate how a mesh incentive-based P2P system can be used for the deployment of a live streaming application. We perform an experimental evaluation of PULSE, an un-

structured live streaming system we designed and developed. We show that *the mesh-incentive approach meets the live streaming requirements and is effective in discriminating peers according to the resources they provide to the system*, giving advantage to the ones contributing more. The analysis of data distribution paths highlights average diffusion properties not far from the design principles of a structured system.

In Chapter 5 we focus on the building blocks of the diffusion process in mesh-based systems: the chunk exchange algorithms. We derive some practically interesting diffusion schemes and analyze their performance by means of simulations and theoretical analysis. We highlight the existence of a natural *trade-off between the diffusion rate and the diffusion delay*, and we show that *some schemes can achieve optimal performance for both metrics*. In heterogeneous bandwidth scenarios, we highlight the importance of *resource awareness* in the peer selection process; nevertheless *a certain level of agnostic selection is needed* for the functioning of the system. We show that a *trade-off arises between the rate-delay diffusion performance of nodes as a function of the level of resource awareness*. We also highlight that some parameters, like the source selection policy, the chunk and probe set size, play a key role for performance optimality.

Chapter 6 concludes this first part.

1.4.2 Video-on-Demand streaming

Chapter 7 prefaces the second part of the thesis, which is devoted to on-demand streaming systems. In such applications the media contents are available in advance and should be stored in the system: this adds a *storage capacity* constraint to the bandwidth requirement of all streaming applications. We present some architectures that have been proposed to deal with constraints of an on-demand streaming service. We analyze their strengths and drawbacks, and we observe that a peer-to-peer based approach can increase the amount of content that could be stored and the number of users the application could serve, but it introduces additional challenges.

In particular, we consider on-demand peer-to-peer streaming systems where there is no central entity, and peers collaboratively participate to store and serve media content. In particular, we focus on algorithms designed for *controlled environments*, like a VoD service deployed in set-top boxes installed at user homes.

In Chapter 8 we analyze the size of the video catalog that VoD systems can provide to users by taking both storage and bandwidth constraints into account. We show that *catalog size scalability can be achieved if peers have an average upload capacity larger than the stream rate of the contents*. This result also holds in heterogeneous scenarios where peers have different upload and storage capacities if some balancing mechanisms are employed.

In Chapter 9 we consider simple content allocation and connection management techniques for the design of video storage and distribution algorithms. We show that *random content allocation is efficient when coupled with caching mechanisms*, and that *dynamic connection management schemes are needed in critical conditions*. Our results are obtained by means of simulations and experimental analysis.

Chapter 10 ends this second part, while Chapter 11 presents a summary of the thesis and an outlook on the work carried out.

1.5 Publications

The contents of this thesis have been partially published in National and International conferences and journals: the results of Chapter 2 are presented in [140]; the contents of the first part are presented in [143, 144, 145] (Chapter 4) and in [141, 134] (Chapter 5), while the second part is presented in [142, 136] (Chapter 8) and in [133] (Chapter 9).

During the thesis we have also considered multi-path routing. The outcomes of this work are not reported in this manuscript and are presented in [135, 139, 137, 138].

Chapter 2

Bandwidth bounds on the performance of peer-to-peer networks

In a peer-to-peer network, peers share part of their resources to realize an application that cannot be provided by a single server because of scalability concerns. Resources can be physical, like network bandwidth or storage capacity, or even logical, like services or contents. In any case the sum of resources consumed by peers at any time, is at most equal to the sum of resources they provide: this is the *conservation law*.

The total amount of available resources is the critical constraint limiting the performance a given system can achieve. Of course, the way these resources are exploited by the system plays a key role in the final performance. Algorithms are designed to use them as better as possible according to their applications. However, they cannot overcome the feasibility bounds imposed by resources.

The *network access bandwidth* is a critical resource for both on-demand and live streaming systems. As all other resources, it obeys to the conservation law: *the sum of data uploaded by peers is equal to the sum of data they download*. This is the **bandwidth conservation law**, which is very similar to Kirchhoff's Current Law. In this chapter, we use the bandwidth conservation law to provide a *unified model* that can describe the performance bounds of most bandwidth-consuming applications including, but not limited to, live and on-demand streaming, and file-sharing. The goal is not to derive explicit protocols but rather to provide theoretical bounds on the performance systems can achieve from the bandwidth-budget perspective.

Specific protocols differ on the way the available bandwidth is allocated to peers to realize the considered service: this allocation determines the final performance of the systems. We do not focus on specific systems but we just analyze a case in which the bandwidth is fairly allocated to nodes (flat bandwidth allocation) for static and dynamic node population in Section 2.3, and a case of non-flat bandwidth allocation by means of a simple incentive model inspired by BitTorrent's Tit-for-Tat policy in Section 2.4.

The aforementioned bounds are valid for systems that are in a *stationary regime*. However, for a correct system dimensioning, it is also important to consider the evolution of systems towards their stationary regimes, and to understand the perturbative effects of discrete arrivals/departures. Thus, in Section 2.5 we characterize these *transient states* and we analyze the accuracy of theoretical bounds on stationary regime systems by means of simulations.

Contents of this chapter are a joint work with Farid Benbadis, Nidhi Hegde and Fabien Mathieu and are partially presented in [140].

2.1 Related work

First modeling based on the conservation law dates back to 2004 when Ge *et al.* [44] propose a closed queueing system to provide basic insights on the stationary performance of a general P2P file sharing application. Yang and de Veciana [123] study the service capacity of such systems in both transient and steady state through a branching process model and a Markov chain model. Starting from the key modeling idea of [123], Qiu and Srikant [98] derive a simple fluid model of a BitTorrent-like system to study its steady-state performance. Tian et al. [111] extend [123, 98] to take into account different peer behaviors according to their download status. Differently from the aforementioned works, Guo et al. [45] provide a model to understand the evolution of a BitTorrent system during its lifetime and the relations among multiple torrents on the Internet.

A first model of a BitTorrent-based live streaming system is proposed in [110] where a basic analysis of the sustainable streaming rates is performed. More recently, Liu *et al.* study [64] the performance bounds of a peer-assisted live streaming system and exhibit trade-offs between different system parameters, namely tree depth, server load, and degree. They focus on bounds of the aforementioned metrics by assuming the system is feasible.

As concern on-demand streaming systems, Parvez *et al.* [87] use the approach proposed in [98] to analyze the impact of the piece selection scheme when streaming a video.

Differently from these works, we do not focus on specific protocols but we consider different applications under two different bandwidth allocations. Moreover, we consider arbitrary peer upload capacity distributions and we derive download performance and conditions for the feasibility of the systems.

2.2 Model

We consider a hybrid peer-to-peer system where nodes collaborate to realize a given service or application (i.e. live streaming, video-on-demand streaming, file sharing and so on). We suppose a given node belongs to one of these three categories:

Leechers The term is inspired by BitTorrent vocabulary and refers to nodes that are actually using the service. For example, peers downloading a file in a file sharing application or playing a multimedia content in a streaming service.

Seeders This term is also inspired by BitTorrent vocabulary and indicates peers that are currently not using the service, but are only providing resources to the system. For instance, peers sharing a fully downloaded file in a BitTorrent session.

Servers Some extra nodes devoted to the service. For example, servers introduced by the service provider into system to increase its resources.

u_n	Available upload bandwidth of peer n
$d(l)$	Download rate of leecher l
r	Stream rate (if any)
k	File size (if any)
λ	Arrival intensity
$p(u)$	Upload bandwidth distribution
N_L (resp. N_S)	Number of leechers (resp. seeders)
T_L (resp. T_S)	Leeching (resp. seeding) time
U_X	Total upload capacity of population X
N_0	Normalized capacity of E ($N_0 = \frac{U_E}{r}$)
α	average upload/required rate ratio
β	Seeders/Leechers ratio
γ	Tit-for-Tat ratio

Table 2.1 : Table of notation.

Both servers and seeders provide resources to the system without consuming any, but we distinguish the two kinds because their characteristics may be different (for instance servers may be less volatile and have more upload capacity than regular seeders).

We denote as L, S and E the set of leechers, seeders and servers respectively. The number of leechers (resp. seeders) in the system is denoted by N_L (resp. N_S) and may vary over time, while we consider the number of servers is constant and denoted by N_E.

Every node n in L, S, or E has an upload capacity u_n devoted to the service. The total network bandwidth shared by peers in a given set X is denoted as U_X and can simply be computed as $U_X = \sum_{n \in X} u_n$. We denote as $\bar{u_X} = \frac{U_X}{N_X}$ the average upload capacity of peers of set X. In the following (except Section 2.3 where the population size is fixed), we consider a steady-state fluid model where there is a large number of leechers $N_L >> 1$. Under the fluid model assumption, it is convenient to express the upload capacities of nodes belonging to a set X as a probability distribution function $p_X(u)$. Then the total and average bandwidth U_X and $\bar{u_X}$ can be defined as:

$$U_X = N_X \bar{u}_X = N_X \int u p_X(u) du. \tag{2.1}$$

Every leecher l achieves a download rate of $d(l)$ while seeders and servers doesn't download any data. We suppose the download capacity of a given leecher may be limited to $d_{\max}$, which we assume constant. Since we model a steady-state regime we consider $d(l)$ of a leecher l is time independent.

Notation is summarized in Table 2.1.

2.2.1 Bandwidth conservation law

The bandwidth conservation law for an hybrid P2P system can be expressed as:

$$\sum_{l \in L} d(l) \leq \min(N_L d_{\max}, U_L + U_S + U_E), \tag{2.2}$$

Equation (2.2) states leechers cannot download faster than the sum of upload capacities of leechers, seeders and servers, or than their physical download speeds.

Note, that Equation (2.2) is only valid for unicast exchanges. If some peers have multicast or broadcast capabilities, the equation should be re-written while taking the leverage effect of multicast/broadcast into account.

In the bandwidth budget we only consider the *effective data transfer*: we do not take into account control messages or other overhead (e.g. transport protocol overhead), and we neglect latency time-shifts.

2.2.2 Performance evaluation

The quality of service (QoS) perceived by users depends on the considered application and, from the bandwidth budget perspective, is only related to their download rate. If the download rate $d(l)$ of a given leecher l is known, it is therefore possible to derive its QoS, or, viceversa, it is possible to derive the required download rate $d(l)$ to assure a certain QoS. In the following, we focus on three applications representative of the most bandwidth-consuming ones:

Fixed rate Applications where the content is generated at a constant rate r and becomes available at a small subset of servers or seeders on the fly. The content is useful only for a short time after it has been generated, and has to be consumed at a rate r. As a consequence a leecher cannot download faster than the generation rate r, because of lack of content, and should download at a rate at least equal to r to assure continuity in content consumption and to respect time requirements. From the bandwidth budget perspective, a measurement of quality of service for such systems is therefore the *continuity* in the content download: the download rate $d(l)$ of a leecher l should be equal to the current content generation rate r at all time. An example of such kind of applications is live streaming.

Required rate Applications where the content is completely available at a subset of nodes and should be consumed at a given rate r. Once again the quality of service can be measured as continuity in the content download. However, in this case, $d(l)$ can also be greater than r because the content is already completely available and it is possible to store or cache it. A typical example of this kind of application is VoD streaming.

Elastic rate Applications where the content is completely available at a subset of nodes and has no restrictions on download time or rate. A leecher l should try to maximize its download rate $d(l)$ in order to minimize its download time, which is the only metric to evaluate the quality of service perceived by a leecher. An example of such kind of applications is file sharing.

By playing with the bandwidth conservation law 2.2, we derive performance bounds of the aforementioned applications. If the system's resources are known, we derive the *download performance* of leechers: this analysis determines the maximal rate a given system can provide to users, and as a consequence the maximal QoS. As a dual problem, we consider a target rate $0 < r \leq d_{\max}$ and we derive the *feasibility conditions* under which this rate can be achieved. This approach is suitable for the dimensioning of live broadcast or video-on-demand services.

In all cases, we suppose the upload bandwidth is *perfectly* exploited by systems' nodes. However, there could be some conditions leading to an under-utilization of the system capacity. This issue is briefly discussed in next section.

2.2.3 Bandwidth dispersion

Peer-to-peer systems are designed to exploit as better as possible the bandwidth shared by nodes in order to provide the desired application or service. In particular, designers should try to maximize the throughput of the system by exploiting all the capacity provided by peers. However, there could be some reasons for an under-use of the available upload bandwidth, leading to a strict inequality in (2.2).

In the following we investigate some conditions that may lead to bandwidth dispersion.

Content starvation Peers may under-utilize their bandwidth because they do not have enough useful data to upload to their neighbors. Peer-to-peer systems are designed to avoid content starvation but distribution algorithms may be inefficient in some scenarios, or limitations impossible to overcome may appear. For instance, this is the case in the early phase of a torrent lifetime, which is called flash-crowd. During this phase, the bottleneck is not the overall upload capacity, but the upload capacity of the initial seeder, which has to inject the first copy of the file in the system [94, 76].

Non-optimal bandwidth allocation A bandwidth allocation algorithm may fail to find an efficient matching between uploaders and downloaders even if such an allocation exists: as a consequence part of the upload capacity is wasted. For example, a node may have not enough downloaders on its neighborhood to fully utilize its upload bandwidth.

Unnecessary bandwidth The application/service may not need all the available bandwidth. For instance, in a live streaming system the bandwidth available for every leecher could be greater than the rate of the stream; as a consequence the extra bandwidth is not used.

Hidden diffusion mechanisms The diffusion to leechers may require side-replication. Consider for instance the fixed-rate scenario described in Figure 2.1. A server $e \in E$ generates a content at a fixed rate r. Three leechers l_1, l_2 and l_3 want to receive the content, downloading it from the server e with the help of two seeders s_1 and s_2. The server e has upload capacity r while other peers have upload $\frac{r}{2}$. Notice that e and the leechers have a total upload capacity of $\frac{5}{2}r$, which is not enough to provide a fixed rate service with rate r to the three leechers (a minimal total bandwidth of $3r$ is required). A solution, shown in Figure 2.1, is to split the stream into three substreams r_1, r_2 and r_3 with upload rate of $\frac{r}{4}$, $\frac{r}{4}$ and $\frac{r}{2}$ respectively. Using s_1 and s_2 as re-transmitters for r_1 and r_2, r can be streamed to all leechers, with an upload cost of $\frac{7}{2}r$. In this example, the diffusion mechanism induces an upload overhead of $\frac{r}{2}$, which corresponds to transfers towards seeders.

We assume content availability is not an issue, we do not take into account control overhead or other implicit overheads (e.g. transport protocol overhead), and we consider that the effect of hidden diffusion mechanisms is negligible. Regarding the latter, it can be shown that the relative overhead can be arbitrarily small if N_S, N_L and the number of substreams are high enough [107]. Since we suppose the bandwidth is perfectly exploited, Equation 2.2 will be an equality unless bandwidth is over-provisioned.

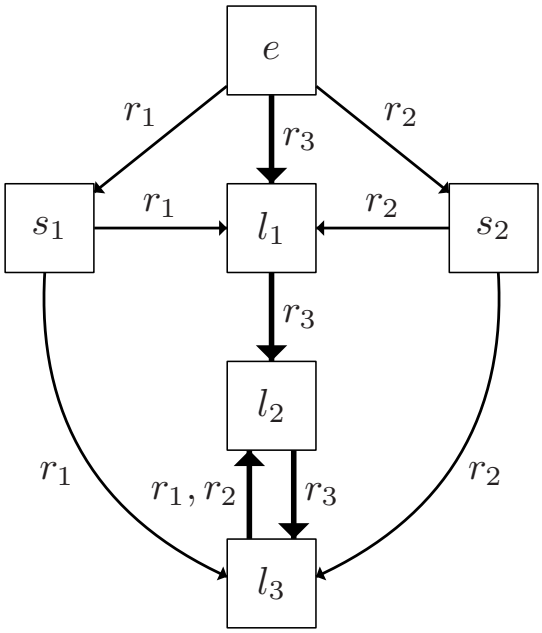


Figure 2.1 : Example of hidden diffusion mechanism. The three leechers need auxiliary seeders to get the service from e.

2.3 Flat bandwidth allocation

In our model the only difference between protocols is the way the available bandwidth is allocated to leechers to provide them the considered service. In fact, we do not take into account control or transport overhead, bandwidth dispersion and latency time-shifts.

In this section we consider the *uniform (flat) bandwidth allocation*, which means that the entire available bandwidth is equally split among all leechers (d is the same for all leechers). This is the simplest possible allocation that can be achieved when each uploader splits its own bandwidth among all leechers. However, it is unrealistic to suppose a peer can upload to all leechers: it is instead more realistic to assume that each uploader chooses at random a few peers to whom it sends content. This results in a good approximation of a flat allocation.

2.3.1 Fixed population

We first analyze the case of a fixed population of nodes: we assume that the number of leechers N_L and seeders N_S are known, as well as their upload capacities U_L and U_S. We also suppose the total amount of resources provided by servers U_E is constant. This models a system for which precise knowledge about the population is available: in this case precise statistics can be obtained for a period of time under which the population does not evolve.

Download performance

By applying Equation (2.2) with uniform allocation, we get:

$$N_L d = \min\left(N_L d_{\max}, N_L \bar{u_L} + N_S \bar{u_S} + U_E\right),$$

which leads to the following download rate for all leechers

$$d = \min(d_{\max}, \bar{u_L} + \frac{N_S}{N_L}\bar{u_S} + \frac{U_E}{N_L}) \tag{2.3}$$

We define as $\beta = \frac{N_S}{N_L}$ the ratio between the number of seeders and the number of leechers of a system. Equation (2.3) can be rewritten as

$$d = \min(d_{\max}, \bar{u_L} + \beta\bar{u_S} + \frac{U_E}{N_L}) \tag{2.4}$$

Equation (2.4) indicates the maximal download rate of leechers when considering a known and fixed population of nodes. Once d is computed it is easy to check if the current system can provide the desired service and which is the QoS perceived by users.

Feasibility conditions

We now consider the dual problem and we suppose a given system targets a rate $0 < r \leq d_{\max}$. The system is feasible for rate r if:

$$r \leq \bar{u_L} + \beta\bar{u_S} + \frac{U_E}{N_L} \tag{2.5}$$

We denote as α_L (resp. α_S) the ratio $\bar{u}_L/r$ (resp. $\bar{u}_S/r$) that indicates the percentage of bandwidth a leecher (resp. seeder) can provide with respect to the target application rate. We denote as $N_0 = \frac{U_E}{r}$ the maximum number of clients that servers can withstand on their own without the need of users' upload capacities. From (2.5) we write the following feasibility condition for rate r:

$$\alpha_L + \beta\alpha_S + \frac{N_0}{N_L} \geq 1. \tag{2.6}$$

We distinguish two situations: $\alpha_L + \beta\alpha_S \geq 1$ and $\alpha_L + \beta\alpha_S < 1$.

If $\alpha_L + \beta\alpha_S \geq 1$, we say the considered system is *scalable* because it can handle an *unbounded number of leechers*. In fact, the target rate r can be achieved for any number of leechers without the need of additional servers. Note that the number of seeders must grow according to the number of leechers in order to keep β constant.

A special case is $\alpha_L \geq 1$ where leechers have the necessary bandwidth realize the service and do not need seeders or servers. Remark that some live streaming [18] and video on demand [142] solutions are already available for $\alpha_L \geq 1 + \epsilon$.

In the special case of $\alpha_S = \alpha_L := \alpha$ the scalability condition simplifies to $\alpha \geq \frac{1}{1+\beta}$; this happens for instance if seeders and leechers have the same upload distribution. We define as $a : \frac{1}{1+\beta} = \frac{N_L}{N_L+N_S}$ the *activity ratio* of the system: in fact it represents the percentage of users that are actually exploiting the service on the total population of leechers and seeders. In this special scenario, the scalability condition is simply that the average relative upload capacity must be greater than the activity ratio:

$$\alpha \geq a. \tag{2.7}$$

On the other hand, if $\alpha_L + \beta\alpha_S < 1$, then the system can only serve a bounded number of leechers and the feasibility condition becomes

$$N_L \leq \frac{N_0}{1 - (\alpha_L + \beta\alpha_S)}. \tag{2.8}$$

We say the corresponding systems are *not scalable* because they cannot provide a service at rate r to an unlimited number of leechers (2.8). However, it is possible to notice that the capacity of servers N_0 is leveraged by a factor $\frac{1}{1-(\alpha_L+\beta\alpha_S)}$. Therefore by using P2P or peer-assisted techniques, a content provider can reduce the server resources needed to handle a given number of users.

2.3.2 Dynamic population

We now consider the population is no longer fixed but the number of leechers and seeders (N_L and N_S) fluctuate according to a random arrival/departure process. We suppose the number of servers N_E is constant and does not vary over the considered period of time. We still consider the available bandwidth is uniformly split among leechers.

Our approach follows [98], but it is more general because we consider arbitrary upload distributions while Qiu and Srikant only consider one class with homogeneous upload capacity, and our model does not target file-sharing systems only. As the systems we describe are more complex, we only consider their steady-state behavior so that the arrival process to all states (leechers or seeders) has the same intensity λ. The validity and limits of this approach will discussed in Section 2.5.

In details, we characterize the system evolution by supposing that peers join the system as leechers according to a process of intensity λ. The upload capacity distribution of the newcomers is indicated as $p(u)$. A given leecher l remains in leecher state for $T_L(l)$ time units before it becomes a seeder. A seeder s provides generous resources to the service for $T_S(s)$ time units, then it leaves the system. For instance, this is representative of the lifetime evolution of a peer in a BitTorrent swarm: the peer joins the system as leecher, it downloads the content and then it remains connected to the swarm as seeder for a certain period of time. Or, it can represent the lifetime of a peer in a VoD session: it enters the system as leecher; it starts the download and after a while the playout; it finishes the download but it could be still playing out the content so it stays in the system as seeder for a certain period of time. Or again, it can represent a peer in a live streaming system that watches a TV channel for a while, then it stops the play out but it lets the application running for a certain period of time.

The leeching and the seeding time (T_L and T_S) may be related to the considered application and to the peers' characteristics. For a given application the only difference between peers in our model is their upload bandwidth, so we can express T_L as a function of u i.e. $T_L(u)$. Following the same reasoning, we assume that T_S is also a function of the upload rate u i.e. $T_S(u)$. For simplicity, we also assume that the content is of size k. This assumption is suitable for elastic/required rate applications but also for fixed rate ones if the content length is known in advance. All leechers take the same time to download the content because the bandwidth allocation is flat: the expected leeching time is therefore $T_L = \frac{k}{d}$ for all peers independently of their upload capacity. It follows that: $p_L = p$, $\bar{u}_L = \bar{u} := \int up(u)du$.

From Little's Law we can derive the expected number of peers in set X in the steady state as:

$$\bar{N_x} = \lambda \bar{T_X} = \lambda \int T_X(u) p_x(u) du. \tag{2.9}$$

that leads to $\bar{N_L} = \frac{\lambda k}{d}$. Note that our fluid model assumption is only valid for $N_X \gg 1$, which corresponds to $\lambda \bar{T_X} \gg 1$ according to Equation (2.9): the interarrival time should be small with respect to the expected leeching and seeding times.

The upload capacity distribution p_X of set X can also be easily deduced from the upload arrival distribution p and T_X: $p_X(u) = \frac{T_X(u)p(u)}{\bar{T_X}}$. p_X may be a sub-probability if T_X is zero for some values of u, but this issue is easily circumvented by setting the missing probability in u to 0 (peers that do not stay do not contribute to the system). It follows that Equation (2.1) can be rewritten as

$$U_X = \bar{N_X} \bar{u}_X = \lambda \int u T_X(u) p(u) du. \tag{2.10}$$

Download performance

We can now derive the download rate d in a dynamic scenario from (2.3) by simply replacing N_S and N_L by their mean values $\bar{N_S}$ and $\bar{N_L}$. We obtain:

$$d = \min(d_{\max}, \bar{u} + d\frac{\bar{T_S}\bar{u}_S}{k} + d\frac{U_E}{\lambda k}) \tag{2.11}$$

One can distinguish two cases in Equation (2.11). If $\bar{T_S}\bar{u}_S + \frac{U_E}{\lambda} \geq k$, the system is in a *over-provisioning state* where the maximal download rate d_{max} can be achieved; this condition will be further described later. Otherwise, Equation (2.11) becomes

$$d = \min(d_{\max}, \frac{u}{1 - \frac{\bar{T_S}\bar{u}_S}{k} - \frac{U_E}{\lambda k}}) \tag{2.12}$$

Equation (2.12) indicates the maximal download rate of leechers in a dynamic scenario when the system is not in a over-provisioned state. As for the static scenario, once d is known it is easy to check if the current system can provide the desired service and which is the QoS perceived by users.

Feasibility conditions

As for the static scenario we can consider the dual problem of the feasibility conditions for a system targeting a rate $r \leq d_{\max}$.

If the system is in an over-provisioning condition, $\bar{T_S}\bar{u}_S + \frac{U_E}{\lambda} \geq k$, it is *scalable*, because any target rate $r \leq d_{\max}$ can be achieved.

Otherwise, according to (2.12), the bounding condition is

$$r \leq \frac{\bar{u}}{1 - \frac{\bar{T_S}\bar{u}_S}{k} - \frac{U_E}{\lambda k}} \tag{2.13}$$

which leads to

$$\lambda k \leq \frac{U_E}{1-\alpha-\beta\alpha_S}, \text{ with } \begin{cases} \alpha = \frac{\bar{u}}{r}, \\ \alpha_S = \frac{\bar{u}_S}{r}, \\ \beta = \frac{r\bar{T}_S}{k}. \end{cases} \tag{2.14}$$

We say the corresponding systems are *not scalable* because there is a maximal arrival intensity they can support. In any case the capacity servers should provide is leveraged by a factor $\frac{1}{1-\alpha_L-\beta\alpha_S}$. Note that feasibility conditions expressed by equation (2.14) are equivalent to conditions expressed by (2.8) extended to dynamic scenarios.

Over-provisioning condition

Theorem 2.1 presents a sufficient condition for achieving optimal download rate $d_{\max}$; note that the condition is easily translated to fixed-rate scenarios by replacing $d_{\max}$ by a given target rate r. This condition is not limited to flat allocation, but works for any efficient allocation scheme[1].

Theorem 2.1 ([140]) *A sufficient condition for achieving $d_{\max}$ with any efficient bandwidth allocation scheme is*

$$\bar{T}_S\bar{u}_S + \frac{U_E}{\lambda} > k(1 - \frac{\bar{u}}{d_{\max}}). \tag{2.15}$$

Two immediate corollaries of Theorem 2.1 are:

- if $\bar{T}_S\bar{u}_S + \frac{U_E}{\lambda} \geq k$, then any finite rate can be achieved,
- if $\bar{T}_S\bar{u}_S \geq k$, then any finite rate can be achieved independently of λ and U_E.

Remark

The last condition is obviously verified if the system verifies $T_S(u) \geq \frac{k}{\bar{u}}$, because we then have $\bar{T}_S\bar{u}_S \geq \frac{k}{\bar{u}}\int up(u)du = k$. In other words, the download is optimal if all peers seed at least the time needed to get the file at a speed corresponding to the newcomers' average upload capacity. If no peer uploads at speed 0, $T_S(u) \geq \frac{k}{u}$ is also a sufficient condition: we get $\bar{T}_S\bar{u}_S \geq k\frac{u}{u}\int p(u)du = k$. In that case, the required condition is that each peer seeds at least the time needed to get the file at its own upload capacity. More generally, for any $0 \leq \mu \leq 1$, $T_S(u) \geq \mu\frac{k}{\bar{u}} + (1-\mu)\frac{k}{u}$ is a sufficient condition in absence of free-riders.

Proof : In the steady state, leechers arrive and leave with the same intensity λ. As any leecher downloads a quantity k between its arrival and its departure, the fluid limit of the total bandwidth $\sum_{l\in L} d(l)$ used by L is equal to λk. As the allocation scheme is efficient, if $U_L+U_S+U_E > \lambda k$, then the download is necessarily optimal. In particular, we have:

- $U_L = \lambda\bar{T}_L\bar{u}_L = \lambda\int uT_L(u)p(u)du$. As the download is limited by $d_{\max}$, we have $\forall l \in L, T_L(l) \geq \frac{k}{d_{\max}}$. Thus we have $U_L \geq \lambda\int u\frac{k}{d_{\max}}p(u)du = \lambda\frac{k}{d_{\max}}\bar{u}$;
- $U_S = \lambda\bar{T}_S\bar{u}_S$.

[1] An allocation scheme is efficient if either the download bandwidth or the upload bandwidth is fully exploited.

It follows that $U_L + U_S + U_E \geq \lambda \frac{k}{d_{\max}} \bar{u} + \lambda \bar{T}_S \bar{u}_S + U_E$, so if Equation (2.15) is verified, the download is necessarily saturated.

□

2.3.3 Applications of theoretical bounds

In this section we propose two examples of how the bounds derived in the previous sections may be exploited for the design and the evaluation of real streaming applications. For simplicity, we consider statistics for systems where the population is fixed; however these examples can be extended to the dynamic case by taking the arrival intensity into account.

Activity ratio and geographical smoothing

The feasibility condition expressed by equation (2.7) states that the relative required bandwidth in a scalable system must be greater than the activity ratio a. In a live streaming service, a is the ratio between the number of users that are actually watching a content and the total number subscribers of the service.

We consider the activity ratio within the Orange France digital television over IP (IPTV) reported in Figure 2.2(a). These values are based on data collected from February 4 to February 10 2008 and aggregated on a typical 24-hour day. From the figure, it is possible to notice that the maximum activity rate is 53%. This value gives a *lower bound on the relative upload bandwidth* required for providing the IPTV service if a P2P or peer-assisted architecture is used.

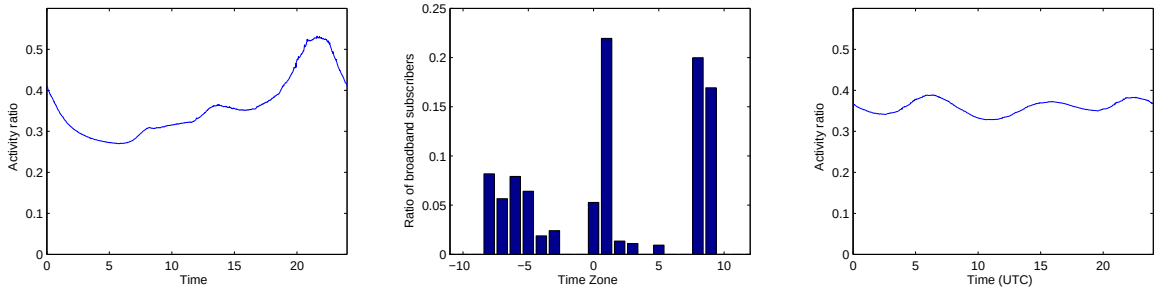
However, this activity rate exceeds 50% during only 2 hours per day, while the average activity is only 36%. A natural question is: can we lower the required relative capacity from the maximum to the average rate?

One solution would be to proactively load the content during hours of low activity. Such a solution requires to have a priori knowledge of which content will be required, and is inapplicable to live streaming content delivery.

On the other hand, if the service is proposed at a worldwide scale, then we have a natural smoothing of the activity because peak hours do not occur simultaneously across time zones. For example, when demand is the highest in Europe (between $8:30$ pm and $10:30$ pm), the service may use seeders from other regions to provide additional bandwidth. Formally, if $a(t)$ designates local activity, which we suppose independent from the area, and P the distribution of users per time zone, then the overall activity A is the convolution of a by P:

$$A(t) = \sum_f a(t+f) \times P(f).^2$$

[2]This not the standard way of expressing a convolution, as the generic term of a convolution is commonly $x(t-f)y(f)$. The reason of our notation is the way time zones are represented, that express the difference between the target time zone and the reference time zone, so that it is possible convert the reference time zone into the target time zone. This is the reverse of the standard notation used in mathematics for expressing temporal shifted events which converts the target time zone into the reference time zone.



(a) Activity during a typical day. (b) Broadband subscribers distribution. (c) Worldwide activity (extrapolation).

Figure 2.2 : Local activity and worldwide extrapolation during a typical day.

Figure 2.2(b) shows the global distribution of broadband users per time zone [1]. Assuming that P follows this distribution, the overall activity is shown in Figure 2.2(c). As convolution always smoothes the original curves, the maximum activity (and hence the relative required upload capacity) is now less than 39%, which is close to the minimum possible value (36%), and much lower than the 53% observed at a local level.

The geographical smoothing allows a non-negligible gain, but it uses physically distant links in order to lower the required bandwidth. This may indeed lead to high latency and overloaded transcontinental links, which goes contrary to the current trends trying to improve locality of data exchange.

Understanding Joost

Joost [55] is a service allowing users to watch movies, videos and TV over the internet. In its original design Joost *was* a peer-assisted system. Joost users had to install a dedicated client which behaved like a leecher (downloading and uploading contents) when a user was watching a video, and like a seeder when the client was idle. However, Joost had recently changed the way the multimedia content is delivered moving towards a traditional client-server approach [16]. In the following we try to explain possible reasons of this change.

A measurement study performed by Hall, Piemonte and Weyant [48], shows that the original version of Joost uses 700 Kbps downstream when leeching, and in average 120 Kbps upstream when leeching or seeding, leading to a relative upload capacity $\alpha \approx \frac{1}{6}$. In fact, the study indicates that about two third of the stream comes from dedicated servers. From (2.6), we deduce that β should be approximately 1 (there are about as many seeders as there are leechers) to provide the target rate of 700 Kbps, and that the system was *non-scalable* at the time of the study. Note that no correlation between geographical location and transferred data was observed, suggesting geographical smoothing.

From Equation (2.7) we observe that, in order to be scalable, the system needs a lower activity a or a higher relative upload α . The former can be obtained by relaying on geographical smoothing: this may be tempting for Joost designers because it costs them nothing, but this will burden the core of the network. Another way to lower the activity is to enforce the seeding behavior. This can be done by proposing strong seeding incentives (for instance reduced commercials).

If the activity is not lowered, with the current measured $\beta \approx 1$ (or equivalently, $a \approx \frac{1}{2}$), the

required α is at least $\frac{1}{2}$, corresponding to 350 kbps upstream devoted to Joost. This represents a large bandwidth for today's DSL connections, which generally offer 1 Mbps upstream. The scalability upload cost may be lower with the new optical fiber connections, and perhaps the designers of Joost hope that the development of their service will coincide with an increase in access bandwidth.

A possible reason of the Joost architectural shift may be this amount of upload capacity required at every user, which is significant for the current deployment of access technologies. However, other non-technical reasons could be the real motivations of this change. For instance a client-server architecture makes possible to watch a stream without installing any client, leading to a simpler service utilization.

2.4 Non-Flat bandwidth allocation: Tit-for-Tat

We now consider systems where peers may have different download rates. This happens when the peer selection process is not random but aware of neighbors resources or contributions. The non-flat rate allocation can also be a consequence of factors non-related to the resource allocation algorithms: different maximal download capacities, number of applications running on the same host, RTTs among peers and so on.

In this section we assume that the *over-provisioning* condition does not hold, otherwise Theorem 2.1 applies. Moreover, we do not analyze the case of a target rate r because this will bring us back to results presented in Section 2.3. In fact, the download rate of peers should be flattened to r and the extra bandwidth devoted to some peers should be reallocated to others.

As a significant example of non-flat resource allocation we focus on *Tit-for-Tat algorithms*, implemented in many file-sharing and streaming applications [30, 41] [144]. These algorithms lead leechers to share their bandwidth preferentially with those from whom they download the most. Therefore the download rate d of a given peer is partially determined by its upload capacity u.

It is difficult to completely describe the behavior of Tit-for-Tat exchanges, especially if peers are dynamic. However, it is generally assumed that Tit-for-Tat can be modeled by a parameter γ, $0 \leq \gamma \leq 1$ [126, 40]. Each leecher shares a fraction γ of its upload bandwidth according to its download history. The remaining $1 - \gamma$ follows a flat allocation, as for the sources and seeders.

We can then propose a formulation of the download rate $d(u)$ under Tit-for-Tat allocation and *fixed population*

$$d(u) = \gamma u + (1 - \gamma)\bar{u}_L + \beta\bar{u}_S + \frac{U_E}{N_L}. \tag{2.16}$$

and under Tit-for-Tat allocation and *dynamic population*

$$d(u) = \gamma u + (1 - \gamma)\bar{u}_L + \frac{\bar{T}_S}{\bar{T}_L}\bar{u}_S + \frac{U_E}{\lambda\bar{T}_L}. \tag{2.17}$$

These equations can be solved numerically through an iterative process. They are valid for elastic and required-rate applications, and for $\gamma = 0$ they are equivalent to Equation (2.11).

In the following we consider dynamic population and we suppose the content is of size k. We can therefore derive solutions of equation (2.17) for some simple and practical scenarios.

2.4.1 Leechers only systems

In this section we analyze systems without external servers ($U_E = 0$) and where leechers leave as soon as they finish their download ($T_S = N_S = 0$). Such systems have been proved feasible, for instance in BitTorrent, after a first copy of the content has been injected into the system: a swarm of leechers can then work without any seeder if the arrival intensity is high enough ([74, 76]).

Without seeders or sources, Equation (2.17) implies that $T_L(u) = \frac{k}{\gamma u + (1-\gamma)\bar{u}_L}$: the service of a peer only depends on u and $\bar{u}_L$, so to find $\bar{u}_L$ solves the system.

Using Equations (2.9) and (2.10), we derive that $\bar{u}_L$ must be the solution to the following equation:

$$\int \frac{\bar{u}_L p(u)}{\gamma u + (1-\gamma)\bar{u}_L} du = \int \frac{u p(u)}{\gamma u + (1-\gamma)\bar{u}_L} du \tag{2.18}$$

Equation (2.18) can be solved numerically for any value of γ, as long as a steady state exists (cf Section 2.4.2). For the limit values of γ, the solution can be explicitly derived:

- $\gamma = 0$ corresponds to the flat bandwidth allocation and implies $\bar{u}_L = \int u p(u) du = \bar{u}$, $d(u) = d = \bar{u_L}$ and $T_L(u) = \frac{k}{\bar{u}_L}$. The mean upload rate of leechers corresponds to the mean upload rate of newcomers, and it is the common download speed.
- If $\gamma = 1$, then each peer downloads at its own upload rate so that $T_L(u) = \frac{k}{u}$. In this case $\bar{u}_L$ is the harmonic mean of the upload rates of newcomers and $\bar{u_L} \leq \bar{u}$.

More generally, by increasing γ, $\bar{u}_L$ should decrease, and the mean leeching time $\bar{T}_L$ should increase: faster peers leave sooner and the service cannot exploit their upload capacities for a long time.

Numerical evaluations

We numerically solve Equation (2.18) for two different peer upload capacity distributions: Gnutella Users and Uniform. The former is derived from a measurement study of Gnutella users [101] while the latter is a uniform distribution on a $[0, u_{\max}]$ range. $u_{\max}$ is set such that both distributions have the same average bandwidth.

In details, we analyze the leecher mean upload capacity $\bar{u}_L$ as a function of the Tit-for-Tat parameter γ. The results are presented in Figure 2.3.

We numerically confirm what stated above: $\bar{u}_L$ is the arithmetic mean for $\gamma = 0$ and the harmonic mean for $\gamma = 1$. Also our intuition is confirmed: as the Tit-for-Tat incentive mechanism increases, faster peers finish their download earlier and contribute to the system for shorter periods of time.

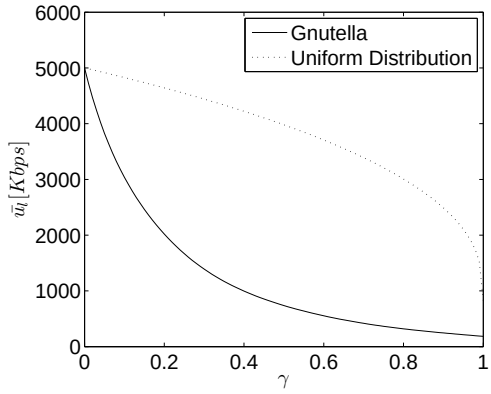


Figure 2.3 : Average upload rate as a function of γ for two distinct upload distributions.

The plot also shows that the uniform distribution is less affected by γ than the Gnutella distribution. This may be due to the high dispersion of the Gnutella distribution. In any case, the conclusion is that the solution of Equation (2.18) is highly sensitive to the bandwidth distribution.

Two bandwidth classes

If we consider a bimodal bandwidth distribution, where arriving peers have an upload capacity u_1 with probability p_1, and an upload capacity u_2 with probability $p_2 = 1 - p_1$, Equation (2.18) becomes a quadratic equation:

$$(1-\gamma)\bar{u}_L^2 + ((\gamma - p_1)u_1 + (\gamma - p_2)u_2)\bar{u_L} - \gamma u_1 u_2 = 0 \tag{2.19}$$

This equation can be easily solved for $0 \leq \gamma \leq 1$, $u_1 \geq 0$,$u_2 \geq 0$.

In the special case where $u_2 = 0$ (a part of the leechers, the *free-riders*, do not contribute at all to the system), the solutions of Equation (2.19) are $\bar{u}_L = \frac{p_1 - \gamma}{1-\gamma} u_1$ or $\bar{u}_L = 0$. In particular, for $p_1 \leq \gamma$ (or equivalently, $p_2 \geq (1-\gamma)$), $\bar{u}_L = 0$ is the only solution which makes sense. This critical value in the bimodal case has been already proved in [126]. We propose in the next section to extend this result to the general case.

2.4.2 Tolerance to free-riders

As stated above, a free-rider is a leecher that is not providing resources to the service (i.e. $u = 0$) but is just exploiting it. In the following, we consider a system where a proportion p_f of leechers are free-riders. The condition for a steady state to exist in presence of free-riders is given by the following theorem:

Theorem 2.2 ([140]) *We assume that $\gamma u \leq d_{\max}$ for all u in the system* [3] *. Then a necessary condition for the system stability is:*

$$p_f \leq (1-\gamma) + \gamma \left(\frac{\bar{T}_S \bar{u}_S}{k} + \frac{U_E}{\lambda k} \right) \tag{2.20}$$

[3] Otherwise, the bandwidth received by a peer through tit-for-tat exchanges should be limited to d_{max}, and the remaining part redistributed to the other nodes.

A steady state exists if Equation (2.20) *is strictly verified and the allocation of the bandwidth not distributed by the Tit-for-Tat mechanism is flat.*

Proof : In the steady state, if any, the total bandwidth used is λk, and the bandwidth used by free-riders is $p_f \lambda k$. For the steady state to exist, a corresponding upload capacity must be available for the free-riders. Equation (2.17) shows that regular leechers download at least at speed γu so that for them $T_L(u) \leq \frac{k}{\gamma}$. This leads to

$$U_L = \lambda \int_{u>0} u T_L(u) p(u) du \leq (1 - p_f) \frac{\lambda k}{\gamma}. \tag{2.21}$$

Because of the Tit-for-Tat policy, the proportion of U_L allocated to free-riders cannot be more than $1 - \gamma$. By comparing the needed and maximum available bandwidth for free-riders, one get the necessary condition

$$\begin{aligned} \lambda p_f k &\leq (1-\gamma) U_L + U_S + U_E & (2.22) \\ &\leq (1-\gamma)(1-p_f)\frac{\lambda k}{\gamma} \\ &\quad + \lambda \bar{T}_S \bar{u}_S + U_E, & (2.23) \end{aligned}$$

which leads to Equation (2.20).

As p_f tends to its critical value, the proportion of free-riders tends towards 1 among the leechers population. If non-Tit-for-Tat allocation is flat, then all the generous bandwidth of the leechers (i.e. $(1-\gamma)U_L$) tends to be redistributed to free-riders, as is the bandwidth of seeders and servers. Thus the regular leechers' download speed tends to γu, and the bandwidth devoted to free-riders becomes arbitrarily close to $(1-p_f)(1-\gamma)\frac{\lambda k}{\gamma} + \lambda \int u T_S(u) p(u) du + U_E$. This retro-action mechanism ensures the existence and stability of a steady state if p_f is strictly below its critical value.

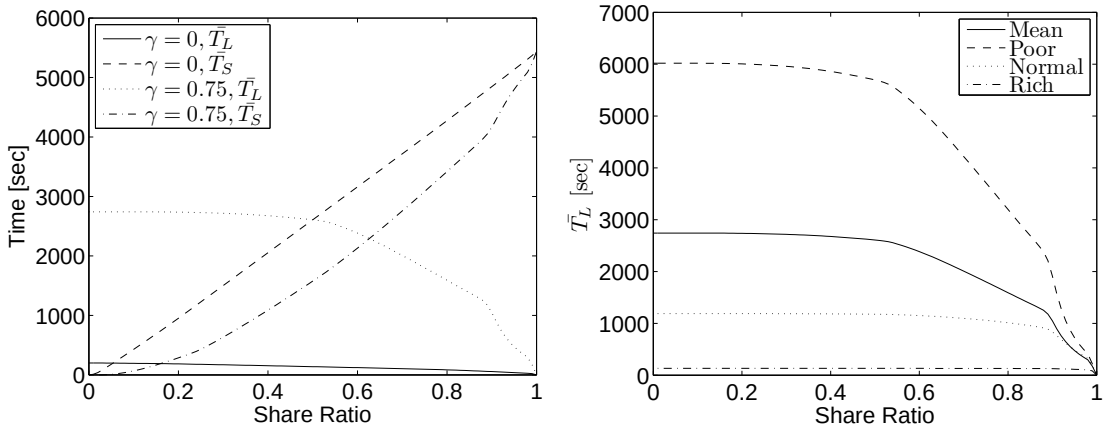
□

If p_f is above its critical value, Theorem 2.2 implies that the system cannot be stable. In practice, the number of free-riders continuously grows because their arrival rate is too large with respect to their download rate. Note that, even under these conditions, the number of regular leechers is stable: they download at speed γu[4].

As a simple illustration, if we suppose $\gamma = 0.75$ (BitTorrent's default parameter) and $N_S = N_E = 0$, the system is able to tolerate up to 25% of free-riders.

The free-riders problem in BitTorrent has already been considered by Yu *et al.* [126]. They propose dynamic resource allocation that depends on the nature of peers (free-riders or not), and show in a bimodal case that BitTorrent's built-in mechanisms can effectively handle some free-riders. Theorem 2.2 generalizes the bound on the number of free-riders a BitTorrent-like system is able to handle by considering arbitrary peer upload capacity distributions.

[4]In real systems, the overcrowded population of free-riders may affect the capacity of regular leechers to efficiently use their Tit-for-Tat schemes



(a) Leeching and seeding time as a function of share ratio for $\gamma = 0$ and $\gamma = 0.75$.

(b) Leeching time (average and per-class) as a function of share ratio for $\gamma = 0.75$.

Figure 2.4 : Impact of the Share Ratio policy on the leeching and seeding time.

2.4.3 Share Ratio policy

The Share Ratio (SR) is the ratio between the amount of data uploaded and the amount of data downloaded by a peer during its stay in the system. It can be considered as a metric to evaluate the contribution of a peer to a service with respect to the resources it exploited. As an example, in BitTorrent communities, it is common to impose a minimal Share Ratio to users in order to improve the performance of the system.

We analyze in this section the impact of such a Share Ratio policy in elastic/required rate systems, by means of numerical evaluations of Equation (2.17). We suppose $k = 100Mb$, $\gamma = 0$ (flat allocation) or $\gamma = 0.75$ (BitTorrent's default setting), and that upload capacity distribution follows the Gnutella Users one. The share ratio SR gives T_S: $T_S(u) = \max(0, \frac{SRk}{u} - T_L(u))$.

Figure 2.4(a) shows $\bar{T}_L$ and $\bar{T}_S$ as a function of the Share Ratio. The share ratio improves the performance of the system by reducing the leeching time $\bar{T}_L$. This is due to the increase of $\bar{U}_S$ in consequence of the increase of $\bar{T}_S$. For $SR = 0$ leechers leave the system as soon as they finish their download as in Section 2.4.1. $SR = 1$ is a critical value leading to an *over-provisioning state* (see Section 2.3.2). Larger values of SR are not considered because they are not compatible with the existence of a steady state.

It is possible to notice that the Tit-for-Tat mechanism ($\gamma = 0.75$) increases $\bar{T}_L$. This can be explained by considering Figure 2.4(b) where $\bar{T}_L$ and the T_L of the three main bandwidth classes of the Gnutella distribution [5] are depicted. We can observe that $\bar{T}_L$ is mostly affected by the performance of the peers with low upload capacities that achieve longer download times. On the other hand, richer peers rely on γ to achieve shorter download times with respect to the uniform allocation, and they are almost unaffected by SR.

[5] We call these classes Poor, Normal and Rich by increasing upload capacity.

2.5 Simulative analysis

The analytical model used so far for dynamic population is based on the steady-state regime of the fluid model. The arrival rate is assumed to be large enough so that the system converges to the steady-state described by the fluid model assumption. The effects of the initial state, where there are more leechers than seeders, are then assumed negligible, and so are the effects of the discrete arrivals and departures in the steady-state. However, a closer analysis of the evolution of the system towards the steady-state and of these perturbations can provide valuable insights for the dimensioning of a peer-to-peer network.

In this section, we analyze such phenomena by means of an event-based simulator. As for the dynamic scenario considered in the previous sections, we suppose peers join the system as leecher according to a given arrival process of a given intensity. At any time, the bandwidth available in the system is the sum of all peers' upload capacity and it is shared among leechers according to a given allocation scheme. We suppose the content is of size k, so that leechers become seeders when they end the content download, and they leave the system after a given seeding time. Again, this assumption is suitable for elastic/required rate applications but also for fixed rate ones if the content length is known in advance. For simplicity, we do not consider the presence of external servers ($U_E = 0$).

In the following we will analyze, in transient and steady state, the evolution of the number of leechers and seeders (N_L and N_S) over time, as well as the evolution of the time peers take to download the file (leeching time T_L).

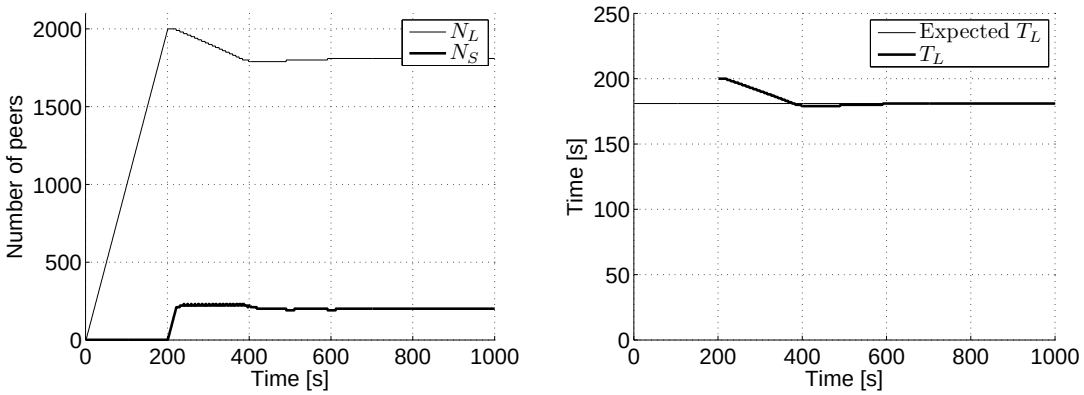
2.5.1 Transient states

We now consider the evolution of the system towards the steady state. In order to clearly exhibit the inherent behavior of the transient states, in the following we report results for a simple scenario: flat (non Tit-for-Tat) bandwidth allocation, homogeneous upload bandwidth u, unlimited download bandwidth, periodic (deterministic) arrivals, and constant seeding time T_S. We assume that a session begins with no seeders except for one initial source of negligible upload, which injects a first copy of the content in the system and leaves as soon as content availability is ensured.

We expect that the system traverses distinct transient phases before reaching the stationary regime:

- As leechers arrive into the system, there is an initial phase where the number of leechers increases linearly and there are no seeders (except the bootstrap source). The leecher population increases at a rate of λ, with a constant download rate per leecher of u. This phase lasts k/u seconds, with about $\lambda k/u$ leechers at the end.

- During a second phase the number of seeders increases as leechers complete their downloads and become seeders. The seeders increase the available bandwidth and shorten the download time so that leechers-seeders transitions should increase (its intensity may be greater than λ for a while). This phase is self-induced as long as the leechers population is not varying too much.



(a) Evolution of the number of leechers N_L and seeders N_S over time.

(b) Expected T_L and evolution of T_L over time.

Figure 2.5 : Transient state $T_S \ll T_L$

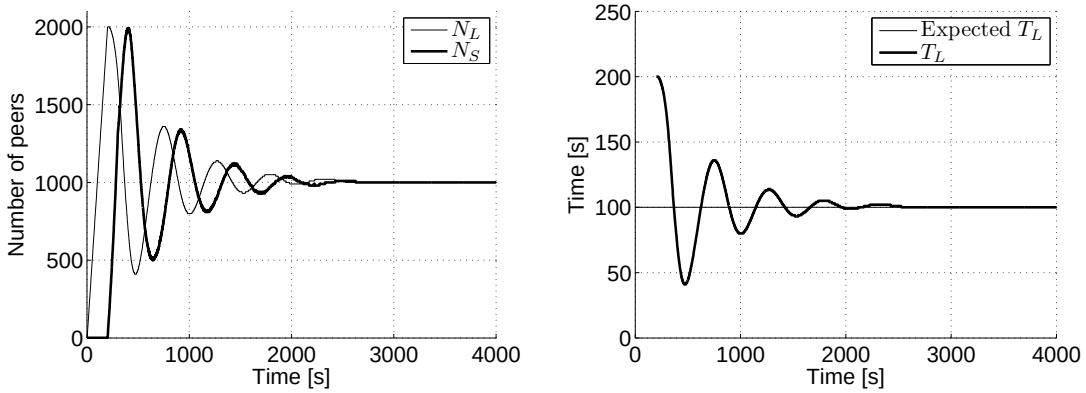
- After some time, first seeders start leaving the system while the longer seeding time cause the leecher population to shrink relatively faster, thus shortening the period of time peers spend as leechers, and consequently after some time reducing the population of seeders. This in turn reduces the global bandwidth available, causing download time and number of leechers to increase. As the leeching time increases, the total time in the system increases and so on...

- The oscillations eventually stabilize after some iterations, and a stationary regime is reached.

We investigate these phases for four illustrative scenarios. The scenarios are categorized according to relative amounts of seeding and leeching time, which indicate the relative strengths of the seeders and leechers population. We define the seeding time T_S as a fraction of k/u. Remember that in the fluid model, for a stationary state without source, we have $T_S + T_L = \frac{k}{u}$ (from Equation (2.11)) in all regimes but the overprovisioned ones 2.4.1. So the various seeding times we consider represent various degrees of balancing between leechers and seeders. Unless otherwise specified, we consider an arrival rate $\lambda = 10\text{s}^{-1}$, an upload rate $u = 5$ Mb/s, and a file of size $k = 1000$ Mb, so that we have a typical time $\frac{k}{u} = 200$ s (download time without seeders) and a maximal leechers population $\lambda\frac{k}{u} = 2000$.

We first consider an *under-seeding scenario*, where $T_S << T_L$, $T_S = \frac{k}{10u} = 20$. Figure 2.5 shows results for the number of leechers and seeders (2.5(a)) and the evolution of T_L over time (2.6(b)). In the first phase, which lasts 200 s ($\frac{k}{u}$), the leecher population increases up to $\lambda\frac{k}{u} = 2000$. This phase is identical in all scenarios because it represents an incompressible phase without seeders. Then we have the second phase, where the seeders population increases. This phase lasts 20 seconds (T_S), until the first seeder leaves the system. Then there is only one oscillation and the stationary regime is attained. Note that the leeching time converges toward the expected $T_L = 180 = \frac{k}{u} - T_S$, and that during the oscillation we have a moment where T_L is slightly smaller than 180.

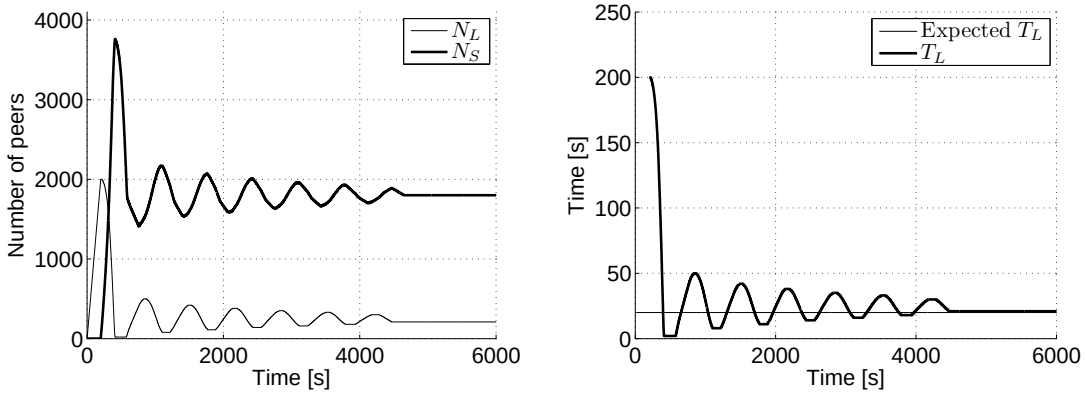
In the second scenario we set the seeding time equal to the theoretical leeching time in the stationary regime: $T_S = T_L = \frac{k}{2u} = 100$ s. Figure 2.6(a) shows some significant oscillations between the moment seeders begin to leave the system and the stationary regime, which is



(a) Number of leechers N_L and seeders N_S over time.

(b) Expected T_L and T_L over time.

Figure 2.6 : Transient state $T_S = T_L$



(a) Number of leechers N_L and seeders N_S over time.

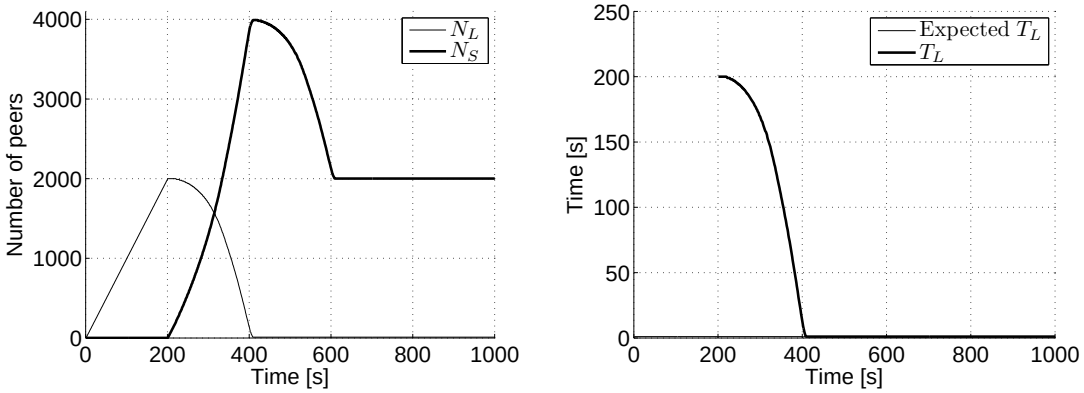
(b) Expected T_L and T_L over time.

Figure 2.7 : Transient state $T_S > T_L$

reached over 2000 s. Under this regime the leeching time $T_L = 100\ s$ and coincides with the expected theoretical one.

The third scenario considers an even larger seeding time, $T_S = \frac{9}{10}\frac{k}{u} = 180$, that is significantly larger than the expected leeching time of 20 s. In this case, as shown in Figure 2.7, the oscillation phase lasts even longer, over 4000 s. The large seeding time exaggerates the oscillations seen in the second scenario leading to a larger convergence time. However, once the system reaches the stationary regime, the leeching time remains constant at the expected value of 20 s. Note that the amplitude of the first oscillation generates a transient over-provisioned state, where the amount of accumulated seeders is (temporarily) so large that the leeching time is smaller than the interarrival time (a leecher can download the whole file before the next leecher arrives).

We finally consider an *over-seeding case*, where the seeding time is set to $T_S = \frac{k}{u} = 200$. Figure 2.8 shows only three transient phases, each one lasting 200 s without extra oscillations: first the growth of leechers, then the growth of seeders while leechers shrink, and last the extra seeders are absorbed. In the following stationary state the download (leeching) time is very small because the seeding time is significantly large; as a consequence the number of seeders is always very large while the number of leechers is always either 0 or 1.



(a) Number of leechers N_L and seeders N_S over time.

(b) Expected T_L and T_L over time.

Figure 2.8 : Transient state $T_S \gg T_L$

To conclude, simulations show transient state is characterized by four distinct phases except in the *over-provisioning* case, where the oscillation phase is not observed because of the large amount of available bandwidth. Moreover, simulations confirm the formulas derived in the previous sections are able to correctly predict the performance of a peer-to-peer network when it is in steady-state. We run simulations with several other parameters, like Tit-for-Tat allocation, heterogeneous upload capacities and so on: in all cases transient state presents the four phases and results confirm our theoretical bounds are correct.

2.5.2 Validation of the fluid model assumption

We now verify the validity of the fluid model in the stationary regime: this assumption has been used to derive performance bounds under dynamic population. In details, we analyze the impact the arrival process, its intensity and the peer upload bandwidth distribution have on the stationary regime performance.

Arrival pattern and intensity

In the simulation results presented so far, we assumed a constant interarrival time, which led N_S and N_L (and thus T_L) to be practically constant in the stationary state. However, the arrival process may not be so deterministic, and we therefore consider a different arrival process. Figure 2.9 shows the impact of a Poisson arrival process for $T_S = \frac{k}{2u} = 100\ s$, that is the existence of permanent oscillations within the stationary regime. We considered three arrival intensities $\lambda = 1, 10, 100$. When the arrival intensity is large, the global bandwidth is large as well and sufficient for the fluctuations to be negligible. When the arrival intensity is small, burst arrivals may find insufficient bandwidth, thus increasing leeching time temporarily and leading to larger oscillations. We thus validate our fluid model assumption for large values of arrival intensity.

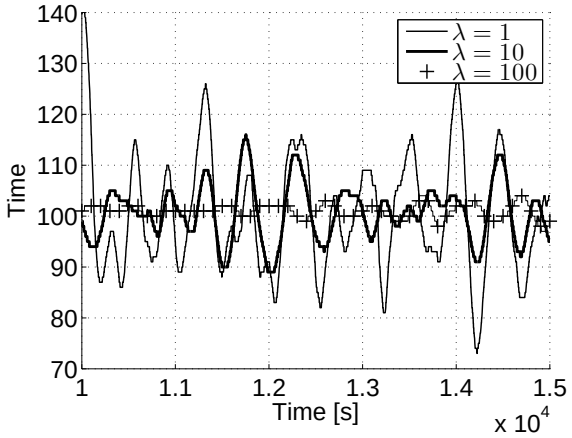
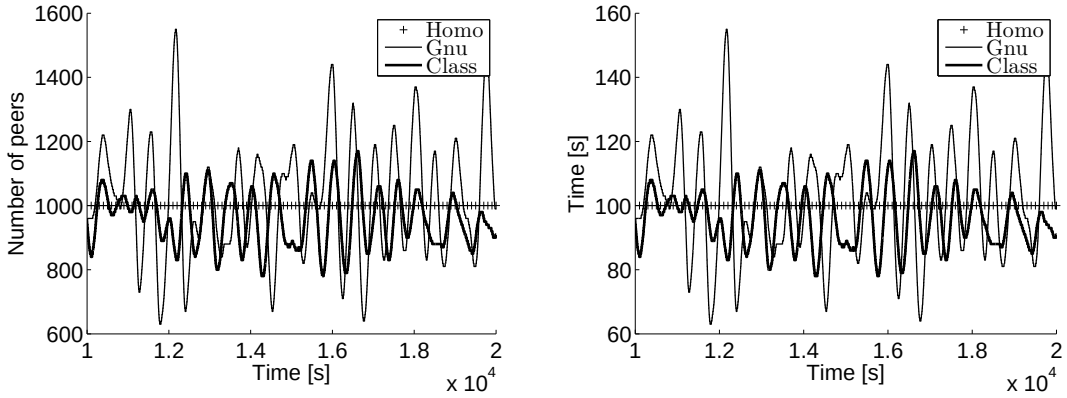


Figure 2.9 : Leeching time over time under Poisson arrivals of different intensity.



(a) Number of leechers N_L over time.

(b) T_L over time.

Figure 2.10 : Impact of the peer upload capacity distribution on the stationary regime performance.

Peer upload capacity distribution

We now investigate the impact of the peer upload capacity distribution on the steady-state regime. In particular, we consider three upload distributions with identical averages: homogeneous bandwidth, heterogeneous will little variation[6] (labeled as Class), and heterogeneous with high variation, as observed in Gnutella systems. In all cases we consider an average upload bandwidth $\bar{u} = 5$ Mb and an arrival intensity $\lambda = 10$. Figure 2.10 reports the number of leechers and the leeching time for these scenarios. The more disperse the upload bandwidth is, the more variable is the available global bandwidth. This variability results in larger fluctuations in the evolution of the number of leechers and leeching time. Since the fluid model considers only the average upload bandwidth, it is more accurate for less scattered upload distribution. In simulations not reported here, we have confirmed that as the arrival intensity increases, the effect of the dispersion in upload bandwidth is less significant and the fluid model is still valid.

[6] 4 classes: 20% at 0.640 Mb, 40% at 1.9 Mb, 25% at 5 Mb, and 15% at 20 Mb

2.6 Conclusion

The performance of a content delivery peer-to-peer network is primarily limited by the total amount of resources shared by participant nodes. In particular, the performance of streaming applications is bounded by the available *bandwidth*, that is the most critical resource for such systems.

In this chapter, we have considered the performance bounds peer-to-peer networks can achieve from the bandwidth budget perspective when they are in a stationary regime. These bounds can be considered as guidelines for the design of the system, the tuning of its parameters and the evaluation of its performance.

In particular, we have derived a unified model, based on the bandwidth conservation law, which describes the performance of a large number of today's popular applications, such as live and on-demand streaming, and file sharing. We have described how, using P2P, a provider can reduce its costs while increasing the capacity of its network, under which conditions a system guarantees an arbitrarily low download time, and which are the download performance of users under a given population. We have also examined the transient phases for various seeding scenarios showing that convergence to the stationary regime can be slow as seeding time increases.

We have also provided some examples of how our analysis can be useful in realistic contexts. For example we have considered the "Joost" case, we have analyzed the impact of share ratio policies and we have derived the maximal number of free-riders a BitTorrent-like system can handle.

Part I

Mesh-based peer-to-peer live streaming

Chapter 3

Introduction

Peer-to-peer overlays have recently become a popular approach for the diffusion of live streaming. Earliest research efforts date back to 2000 when overlay systems have been introduced to reproduce a multicast infrastructure at application layer when no native support for IP multicast was available ([39, 26]). The first approaches to P2P live streaming, such as SpreadIt [12], inherit much from application level multicast. The proposed overlay is a *single-tree* to route data from a source to a population of users. To improve scalability and resilience, NICE [10] organizes nodes in a hierarchical cluster-based single tree overlay, while ZIGZAG [113] improves this design by adding redundancy to the cluster management. EMS [25] has been the first single-tree multicast system to be deployed and for which measurements have been collected. The main limitation of single tree systems is that leaf nodes cannot contribute with their bandwidth to the system. Moreover, internal nodes may become bottlenecks, disrupting the delivery of data to a large number of users.

Multiple-tree based overlays have been proposed to address the aforementioned issues. The stream is encoded in independent sub-streams (often called stripes or description) and forwarded over different trees. CoopNet [85] is the first to introduce this approach, but it still relies on a powerful central server. Splitstream [18] proposes interior-node disjoint trees, where a node is an interior node for only one diffusion tree and a leaf for all the other trees. In this way, the bandwidth of all nodes is exploited and the presence of a bottleneck node or a failure affects only the diffusion of one sub-stream. On the other hand, the overhead to maintain such infrastructure is higher than in the single-tree case, and churn and bandwidth heterogeneity may be troublesome [13]. Other systems followed the multiple-tree approach, such as [41, 102]. The former proposes an original topology to improve tree-building algorithms, while the latter adapts multiple-trees to heterogeneous resource distributions. Recently, Chunkyspread [115] reduces the requirement of a structured substrate by adopting a non-hierarchical approach to tree building.

In contrast with these *tree-based* systems, the *mesh-based* approach reduces the structural requirements making the system more adapted to resource heterogeneity and intrinsically resilient to node churn [69]. This approach has already been showed efficient to deal with these issues in P2P file-sharing systems, such as BitTorrent [14]. In mesh-based live streaming systems the stream is divided in a series of pieces (chunks), that are injected in the system by a source and are then exchanged among peers in order to retrieve the continuous sequence and play out the

stream. Content dissemination is therefore driven by chunk exchange algorithms executed locally, which can be described by their chunk-peer selection policies. Every chunk follows its own path from source to nodes, leading to a different distribution tree for every chunk.

Bullet [57], is an early proposal that combines a single-tree with a mesh, while Chainsaw [86] is the first example of a mesh-only system based on random peer-chunk selection algorithms. Coolstreaming/DONet [130] introduces a better chunk scheduling algorithm that takes into account the chunk playout deadlines, while PULSE, better described in Chapter 4, is based on a BitTorrent-like *Tit-for-Tat* peer selection strategy and local *rarest first* chunk selection policy. Other examples of mesh-based live streaming systems are GridMedia [127], which uses a combination of push/pull mechanisms for stream diffusion, and PRIME [68], which combines multiple diffusion trees and swarming.

In the last years, commercial systems like CoolStreaming [130], PPLive [95], SopCast [105], TVants [114] and UUSee [53] have become increasingly popular. Although they are closed-source applications, it seems that they are all based on a mesh design and relay on chunk exchange algorithms for the stream diffusion. The price to pay for the flexibility of the mesh-based approach is a random, hardly predictable performance. The goal of this thesis part is to analyze the main performance trade-offs of resource allocation algorithms for data dissemination in mesh-based live streaming systems.

Even more recently, commercial systems based on structured approaches have also been deployed, such as Stanford Peer-to-Peer Multicast (SPPM) [82] and Peerialism [88]. A comparison of tree- and mesh-based commercial live streaming systems is proposed in [4]: main results highlight tree-based systems require less overhead, while mesh-based ones provide lower start-up delays and can better mimic the underlying IP network being more network aware.

3.1 Resource allocation in mesh-based live streaming systems

The resources available in a mesh-based P2P live streaming system are exploited to exchange chunks between peers in order to distribute them as fast as possible while guaranteeing playout continuity. Resource allocation algorithms run at every node and are designed to meet these requirements by means of *local decisions* taken on the basis of *local knowledge*. Remember that, in a live streaming application, the most critical resource is the network bandwidth: as a consequence the chunk exchange algorithms define how this bandwidth is used at every node.

A basic knowledge is required to assure *connectivity* between different nodes in the system. Every peer should know a certain number of other peers: we say these nodes are the neighbors of the considered peer and we define the *overlay* as the set of nodes and knowledge links. In a tree-based application this connectivity is implicitly assured by the tree structure itself: the overlay corresponds to data distribution trees. However, every node may eventually know additional nodes to improve robustness. In a mesh-based approach, this implicit structure is missing, so that connectivity can be handle in several ways. For instance, it is possible to use a *structured* solution, where there is an infrastructure, like a DHT or a tree, to which nodes belongs to: the mesh used for data exchange is a subnetwork of this structure. Or it is possible to use an *unstructured* approach, where a node knows other nodes in the systems thanks to the exchange of gossip-like messages without building an explicit structure. It is also possible to use

a *central entity*, like a tracker, that provides to every node a list of neighbors. Or again, it is possible to design solutions that are based on hybrid approaches derived from the aforementioned proposals.

Chunk exchange algorithms run on top of the knowledge overlay and may be broadly categorized as *push* or *pull*, depending on whether it is the *sender* or the *receiver* that does the selection, respectively. Push-based schemes are more suitable for upload-constrained systems, which is representative of peers connected through ADSL or cable for instance, since the dissemination of chunks is regulated by the sender as a function of its upload capacity. Pull-based schemes, on the other hand, are more appropriate to describe download-constrained systems, since the rate of chunk requests adapts to the download capacity of each peer. These allow peers to request those chunks that are closest to their playback deadline.

This classification is however not that clear in practice. In fact, in case of push schemes, it is necessary to avoid collisions, i.e. avoid that two senders select the same chunk for the same peer at the same time, otherwise part of resources are wasted because only one transfer is actually useful. This can be done by proposing a given chunk to a recipient that should acknowledge the decision before the actual chunk transfer, or by detecting the collision at receiver side before the transmission ends. If pull schemes are used, collisions are naturally avoided but a recipient is not sure that the selected sender will provide it the desired chunk. There is therefore the need of timeouts or explicit acknowledges generated by senders. So, in practice, a diffusion algorithm is actually based on an *hybrid push/pull* approach where the chunk exchanges are *negotiated* between peers.

Moreover, under both approaches, the basic connectivity knowledge is not sufficient for chunk exchange. Regular exchange of information about data owned by neighbors is indeed needed by a node to perform the selection. The amount of overhead generated for control exchanges depends on the algorithm. For example, a scheme like *random peer/ latest blind* (Chapter 5) chunk does not require any additional information exchange at all, while schemes based on *rarest first chunk selection* require knowledge of the status of all neighbors. For these lasts the information may be incomplete or not up to date in order to reduce the amount of overhead, but in this case the performance of the diffusion algorithm is worse as well.

Another key parameter is related to *source coding*. In practical P2P live streaming systems, the source contains some redundancy in the original signal to recover from chunk losses or delivery beyond the playback delay. The design of the source coding scheme should depend on the performance achieved by the diffusion scheme and on other parameters like the fairness of the chunk dissemination among peers, the statistical characteristics of successive chunks received by any given peer, and the additional delay introduced by the source coding/decoding algorithms. Source coding can be based on simple Forward Error Correction techniques (like Reed-Solomon), or on more complex mechanisms like MDC or layered coding allowing to provide adaptation of the received media quality to the available resources [66].

Recently *network coding* has been introduced in the live streaming context. Network coding eliminates the need of chunk negotiations because blocks exchanged are linear combinations of stream chunks, and, as a consequence, it allows a better utilization of the available bandwidth. Authors of [118, 119] show the use of Gauss-Jordan elimination techniques allows fast chunk decoding because chunks are decoded while they are received.

Practical diffusion schemes, especially in heterogeneous bandwidth scenarios, should be aware of the resources shared by nodes. In particular, it is important that nodes with higher upload

capacities are placed in the first levels of distribution trees in order to minimize their length and therefore provide small play out delays. Moreover, practical diffusion schemes should be robust to node *cheating* and *selfish behavior*. Schemes that allow a peer can improve its reception rate or delay by sending false information about its state may rapidly collapse. Peers should also be encouraged to upload chunks at the highest possible speed, using for instance some form of *Tit-for-Tat* mechanism similar to that of Bit-Torrent [30].

Recently, increasing attention has been devoted to the design of mechanisms for network-awareness. These mechanisms influence the knowledge overlay building (like [91, 122]) or directly act in the peer selection process for data forwarding (like [144]). Some of them are simply based on latency or bandwidth measurements, while more complex ones rely on external trackers to retrieve topological information [120].

Metrics for performance evaluation

The main goal of a live streaming application is to minimize the play out delay while guaranteeing play out continuity and media quality at receiver nodes (see Chapter 1 for more details). The most important metrics to evaluate the performance of allocation algorithms for such applications are therefore the *diffusion delay* and *diffusion rate*.

The rate/delay performance trade-off achieved by each algorithm is evaluated through the diffusion function r, where $r(t)$ is the probability that it takes no more than t time units for an arbitrary chunk created by the source to reach an arbitrary peer. Equivalently, $r(t)$ is the fraction of peers that receive any given chunk no later than t time units after its creation, averaged over all chunk transmissions.

The diffusion function has the typical S-curve illustrated by Figure 3.1. We refer to the asymptotic value of $r(t)$ as t tends to infinity as the *diffusion rate*. This corresponds to the average fraction of chunks received by an arbitrary peer; equivalently, this is the average fraction of peers that eventually receive any given chunk. In revers, it is possible to define the *chunk miss ratio* (or simply *miss ratio*) as the asymptotic probability to miss a chunk, that equivalently corresponds to the fraction of peers that do not receive any given chunk.

As concern the *diffusion delay* we should distinguish two main metrics: the *average diffusion delay* and the *maximal diffusion delay*. The former is defined as the time needed for a chunk to reach a peer on average. The latter is defined as the delay it takes for an arbitrary chunk to reach a fraction $1-\varepsilon$ of the peers that will eventually receive that chunk, where ϵ is an arbitrary, small constant.

Another important metric to take into account for the evaluation of diffusion schemes is the *overhead*, which is the difference between the bandwidth used by peers (throughout) and the actual data received (goodput). In particular, we can distinguish the overhead related to the maintenance of the knowledge overlay, the overhead needed to negotiate chunk exchanges, and the overhead generated by unnecessary data transmission (like two copies of the same chunk to the same peer).

As already explained, in a mesh-based system every chunk follows a different path from the source to the nodes. An important aspect to analyze to understand the functioning of diffusion algorithms are the *properties of these chunk distribution trees*. Of particular interest are the tree lengths and tree widths and their evolution over time.

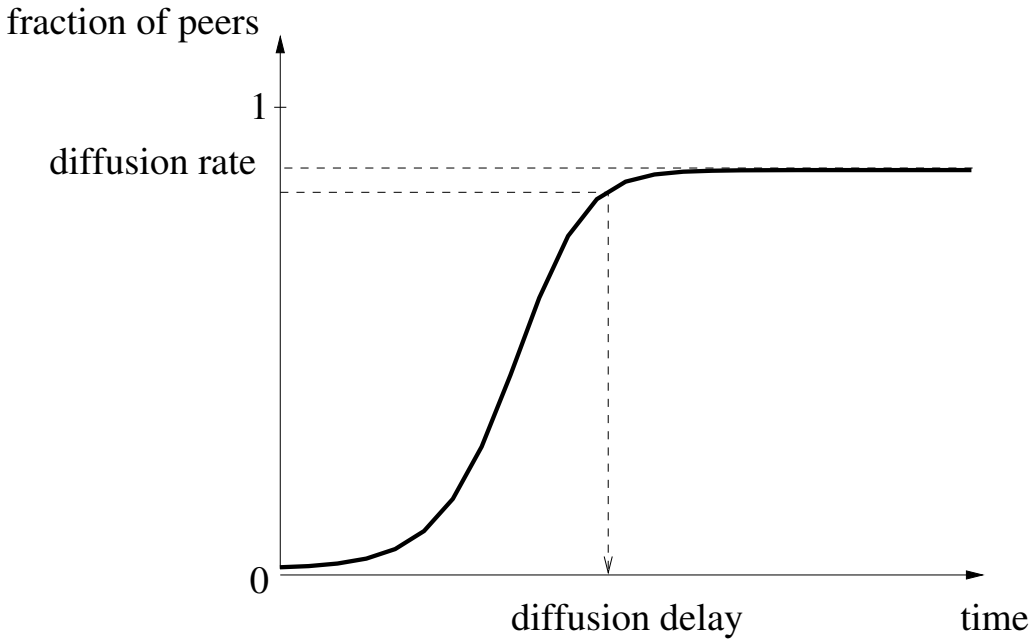


Figure 3.1 : Performance metrics associated with the diffusion function: diffusion rate and diffusion delay.

3.2 Contributions

In this thesis part, we undertake a deep performance evaluation of resource allocation algorithms in mesh-based live streaming systems: these algorithms drive the content dissemination by means of chunk exchange policies.

In Chapter 4 we consider PULSE, an incentive mesh-based live streaming system we designed and developed. Its algorithms are completely known, so that we can fully understand the application behavior from a test-bed evaluation. The two main goals of the chapter are the following: i) to understand whether a mesh-incentive approach can meet the live streaming requirements, and is therefore suitable for the deployment of a live streaming application; ii) to analyze the diffusion performance of PULSE and to understand the fundamental properties of the data distribution paths its algorithms generate. However, from this experimental analysis it won't be possible to analyze all the aspects of the diffusion process and to derive a general model: results are influenced by the design choices of PULSE and by the complexity of a whole developed application.

In Chapter 5 we therefore tackle the problem from a more theoretical perspective and we consider the chunk/peer selection policies only. First, we propose some practically interesting diffusion schemes, we analyze their diffusion rate/delay performance in homogeneous systems where all peers have the same upload capacity, we derive explicit formulas to describe the diffusion function of some algorithms and we verify whether optimal performance can be achieved. We then consider the case of heterogeneous upload capacity, we extend the model to schemes that take into account the resources shared by nodes when performing peer selection (i.e. resource-aware algorithms), and we analyze the main performance trade-offs in such heterogeneous scenarios. Finally, we consider the impact some crucial system parameters have on the performance of optimal diffusion schemes.

Chapter 4

PULSE experimental analysis

A live streaming system targets to distribute a multimedia content to the largest possible audience, while minimizing the playout delay and assuring the quality and the continuity of played content at receivers. If a peer-to-peer approach is used to realize the application, additional constraints should be taken into account. Nodes may be volatile and join/leave the system in unpredictable ways; the resources provided by peers may be different; nodes may be placed everywhere on the underlying network topology and so on. These constraints are particularly stressed if the system is employed in uncontrolled environments, like in the case the application runs on user PCs. The application should therefore assure service scalability, while being aware of resources and locality of nodes, and being resilient to node transiency.

A fixed overlay structure considers *churn* as an exceptional event and performs operations to re-organize the system toward a "normal" state. These operations should be performed as fast as possible to assure correct data delivery to nodes. On the other hand, a mesh structure supposes a certain churning is always present and operates normally under this condition. Moreover, the churning is eventually considered a good characteristic of the system in order to optimize the overlay and improve its stability.

In a structured system nodes are organized in a static hierarchical structure, while a mesh-based overlay is flexible in organizing the *resources* of the system. Every peer is free to manage its connections in order to retrieve/upload the stream. This allows every node to contribute as much as it wishes to the system and the resources are completely exploited. In particular, the peers contributing the more can fully utilize their bandwidth, and, at the same time, the impact of low bandwidth peers is minimized.

This freedom in connection management also allows a dynamic optimization of the overlay according to *content availability*. Every node can look for the content from potentially every node in the overlay, so that the overlay structure is naturally optimized at any time. On the contrary, a structured approach should design explicit mechanisms to optimize distribution trees according to content and resource availability. These mechanisms should act extremely fast to meet the live steaming requirements.

Early experiences with the live streaming system Chainsaw [86], highlight that a fully mesh approach performs as well as a structured one in term of diffusion rate while being almost insensitive to node churn. A simulative analysis presented in [69] confirms that the static mapping of content to a particular overlay tree, and that the diverse placement of peers in different overlay trees are two key factors leading to the inferior performance of the tree-based approach.

This dynamic adaptation of the overlay under the mesh-based approach can easily incorporate *incentive* mechanisms in order to promote nodes to contribute with more resources. These mechanisms should give advantages to peers contributing the more to the system in order to incentive resource sharing. Incentive to cooperation have also been proposed in structured systems where nodes contributing the more can can have a larger number of backup connections [102], or can join more diffusion trees [108].

Incentive mechanisms over a mesh structure have already been shown efficient for the deployment of P2P applications over the Internet. BitTorrent [14] is probably the best example of such kind of systems for a file-sharing application. Nodes are organized in a mesh-overlay and resource allocation algorithms are based on the *Tit-for-Tat* mechanism. Under Tit-for-Tat every peer selects as receiver peers the ones that provided more resource to it. Experimental evaluations of BitTorrent [30, 59] highlight that Tit-for-Tat allows the best resource contributors to associate together and get higher download performance.

In this chapter we propose a mesh approach based on incentive mechanisms, which is suitable for the real deployment of a P2P live streaming application. In particular, we investigate how a mesh-incentive solution can meet the live streaming requirements. To this purpose, we perform an extensive experimental evaluation of PULSE, a mesh-based live streaming system we designed and developed. PULSE resource allocation algorithms are based on *Tit-for-Tat* peer selection and *Rarest first* chunk selection policies. These strategies have been adapted to the live streaming context by taking into account the average stream delay of nodes in peer selection, and by restraining the chunk selection over a time sliding window of chunks. PULSE is open source and is now developed under the Napa-Wine european project [79], and the french project P2PIm@ge [84][1].

Contents of this chapter are a joint work with Fabio Pianese, Joaquin Keller, Ernst Biersack and Fabien Mathieu and are presented in [143, 144, 145].

4.1 System overview

PULSE stands for *Peer-to-peer Unstructured Live Streaming Experiment*; this acronym symbolizes the main characteristics and design principles of the application.

In PULSE all nodes are identical except the source, which differs in that it is the first node to distribute the original stream. An unstructured, randomized gossip membership protocol [43] is used to distribute around knowledge information with low overhead, in order to assure connectivity between participant nodes. In addition to these messages, nodes exchange among themselves more detailed messages containing information on the average stream reception performance. Based on this knowledge and on current measurements, the nodes temporarily associate to exchange the data generating a mesh of data connections.

Resource allocation algorithms for the management of data connections are based on a combination of two incentive mechanisms: an *optimistic tit-for-tat* peer selection policy, and an *excess-based altruistic* incentive. Intuitively, the primary mechanism should foster cooperation

[1]The PULSE code developed by the Napa-Wine consortium can be downloaded at `http://www.napa-wine.eu/cgi-bin/twiki/view/Public/PULSE`, while the code developed in the P2PIm@ge project can be found at `http://p2pimages.devoteam.com/`

among resourceful nodes, while the other should both facilitate peer discovery and allow the richest nodes to contribute more effectively to the system.

The fundamental role of incentives in PULSE *is not to enforce fairness understood as as equality of contribution* between nodes. Instead, incentives are primarily intended to optimize the system structure for better global performance. For instance, resource-rich nodes located near the source are able to serve a large number of neighbors with more recent data: they benefit, in terms of reduced playout delay, and the whole system benefits, in terms of reception performances. If nodes whose resources are scarce were recipients of recent data too often, they could slow down or disrupt the distribution process. Also, since time constraints for live streaming are quite strict, we believe it is more important to fully exploit the available resources than to enforce a strict reciprocal incentive for its own sake.

4.1.1 The stream

The source splits the stream into a series of pieces called *chunks*. At this stage, the source may apply to the data a fixed-rate error correction code [80], to achieve better resilience to chunk loss. Chunks are numbered and marked with their original timestamp (we call this time reference the *media clock*) to allow peers to correctly rebuild the initial stream and estimate their own playout delay. Chunks are then made available to the nodes at a constant rate λ. We assume all chunks are of equal size c. If the stream rate SR is constant, chunks are generated at a rate $\lambda = \frac{SR}{c}$. If the stream rate is variable, the chunk generation rate is $\lambda = \frac{SR_{max}}{c}$: when $SR(t) < SR_{max}$ padding is added to keep the chunk size constant. This choice is made for sake of simplicity: we do not discuss further the design of techniques to deal with variable bitrate because is out of our scope, but we argue better ones may be implemented in a real system.

4.1.2 The lag reference system

In Figure 4.1 we illustrate the fundamental concepts and variables used throughout this chapter. The horizontal axis represents the *lag*, which is defined as the age of a chunk with respect to the current media clock. The *newest* chunks are on the left, while the *oldest* ones are on the very right. Over time, the representation of a given chunk moves from left to right: its lag value grows as new data are encoded at the source and present data become older.

We chose this *differential* reference system because it will ease the representation of the buffer dynamics. For instance, since the playout rate is constant, the chunk a node should be playing at any given moment is described by a fixed lag value, the *playout delay* which we denote as T_V. This notation also allows us to define the range of chunks a node is both interested in receiving and capable to provide at some point in time by two values: the *average lag of the window of chunks the node is interested in*, $T_{B_{avg}}$, and the *lag of the chunk a node is going to discard* from its buffer, T_D. The value of $T_{B_{avg}}$ is the average of recent instantaneous values, $T_{B_{inst}}$, sampled over a fixed time frame. These variables are better described in the next section.

This notation is mainly useful for the phase of peer discovery, when it is important to find nodes that are able to provide useful chunks. We can imagine that, when the system is in a steady state, nodes tend to settle on constant average reception delays. In this situation, to discover a potential partner, it is sufficient to compare at any time the nodes' buffer delay ranges. This

can eliminate the need of continuously sending and requesting updated buffer information on a chunk-by-chunk basis. On the other hand, when the system is not in steady state, nodes' buffer delay ranges can fluctuate. However, the information on the buffer delay range is still much less volatile than the information on single chunks or chunk ranges: in normal operating conditions (i.e. while most nodes manage to retrieve chunks at a sufficient rate on a regular basis), a node's reception delay will typically change quite slowly over time. It is thus still possible to use the buffer delay ranges, within a reasonable time frame after their computation, as an approximate and concise representation of the current buffer content at the remote node.

4.1.3 The peer

A PULSE peer is an application that interfaces with the network to steadily retrieve data chunks and control messages. Its goal is to reconstruct the original stream of media data and to pass it to the software player. Its main components are 1) the *data buffer*, where chunks are stored before playback, 2) the *knowledge record*, where information is kept about remote peers' presence, data content, past relationships, and current local node associations, and 3) the *bandwidth allocation algorithms*, whose role is to request chunks from neighbors and to choose and schedule the ones that have to be sent.

Data buffer

Each node has a buffer in which it collects and stores data chunks prior to playback (Figure 4.1).

The buffer uses a sliding window to regulate the stream reception. The *sliding window* is W_S chunks wide. Its goal is to output a stream of chunks with a desired maximum loss ratio. We call *buffer edge* the left most end of the sliding window. We refer to the sequence of chunks the peer is currently trying to obtain through exchanges with neighbors as the peer's *trading window*. This window is denoted as TW and its size is twice than the sliding window one.

We define by *instantaneous position* of a node in the system, referred to as $T_{B_{inst}}$, the lag of its buffer edge from the source. This value can fluctuate quickly, so nodes keep a running average of their instantaneous position, previously referred to as $T_{B_{avg}}$, to filter the short-term position variability due to the unpredictable delays of the data exchange process. The $T_{B_{avg}}$ value indicates the average age of the chunks in the trading window: we will use this variable as an indicator of the *average diffusion delay* perceived by a node. As above, T_D is the fixed lag after which a chunk can be discarded.

Moreover, we define the *safety margin* T_Q as the interval of chunks ranging from the end of the sliding window to the chunk being currently played. The play-out delay T_V is initially set as $T_V = T_{B_{inst}} + W_S + T_{Q_{init}}$ after enough data chunks (to fill at least the configurable $T_{Q_{init}}$ interval) have been gathered. As $T_{B_{inst}}$ is free to change and since T_V remains constant (until the peer either disconnects or suffers from buffer under-run), T_Q is then equal to $T_V - (T_{B_{inst}} + W_S)$. T_Q's function is twofold: it grants an initial safety margin against variations of $T_{B_{inst}}$over time, and the changes in its size can be used by peers to evaluate their current data-reception stability.

A *sliding tolerance* parameter ST defines the minimum amount of chunks that have to be present inside the sliding window before it can move forward. The *maximum chunk miss ratio*

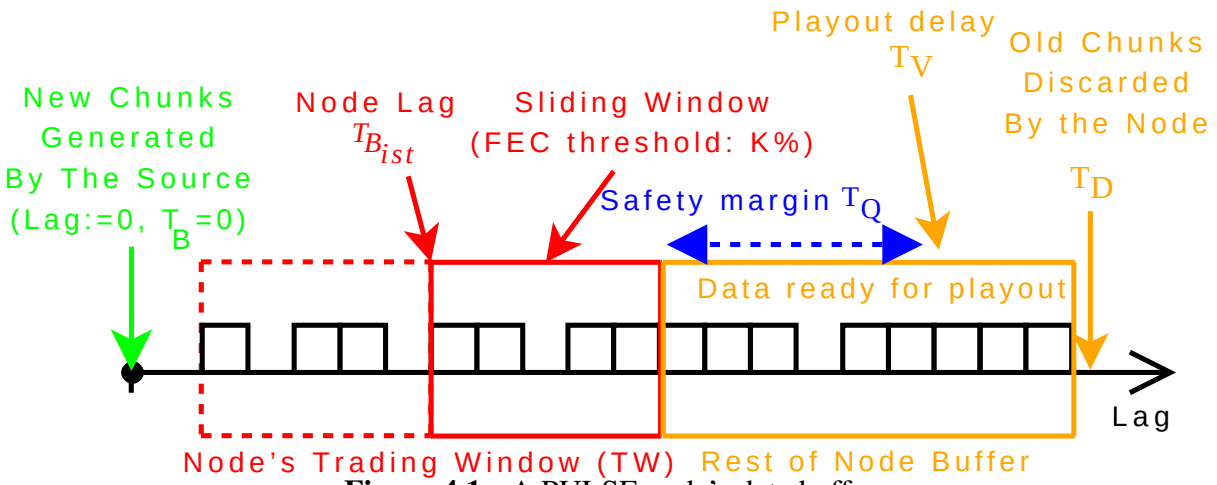


Figure 4.1 : A PULSE node's data buffer

tolerated during normal peer operation is thus bound by $1 - \frac{ST}{W_S}$. The system-wide parameter $miss\ ratio_{max}$ is equal to the amount of redundant coding performed by the source. The value of ST at any peer must be set so that $miss\ ratio \leq miss\ ratio_{max}$ to ensure the complete recovery of the original stream, but peers with enough bandwidth resources can obviously decrease their sliding tolerance at will.

If less than ST chunks are available, the sliding window cannot move. The lag of all the chunks it contains increases as time passes and as new chunks are generated. In this situation, the window keeps drifting on the lag axis (to the right of Fig. 4.1) and $T_{B_{inst}}$ grows at constant speed. Only when at least ST chunks have been collected, the window is allowed to slide and to reduce its $T_{B_{inst}}$ (to the left of Fig. 4.1). The window will then keep sliding as long as it contains at least ST chunks.

When a node joins the system it begins requesting chunks in the interval $[0+\delta, W_S+\delta]$ where δ defines the lag with respect to the source at which peers try to retrieve chunks. This value should be tuned to fall in a range where there are much chunk copies in order to speed up the retrieval of first chunks. When there are enough chunks in the buffer we say the *buffer is connected* and starts its normal operations.

It may happen that T_V, that increases proportionally to the stream rate SR, reaches the right edge of the trading window ($T_{B_{inst}} + W_S$) because data are retrieved at a rate slower than SR. In this case we say there is a *buffer disconnection* where the playback stops, and all the buffer parameters are reseted.

Over time, $T_{B_{avg}}$ will be either decreasing, if the window is sliding at a higher average speed than the source generates chunks, or growing, if the window is stuck waiting to fill a gap or sliding with a lower speed.

Knowledge management

In PULSE the neighborhood management and the propagation of detailed information about data owned by peers are handled by means of unstructured mechanisms.

The strict timing constraints on the data retrieval process emphasize the central importance of the concept of node position in the system that carries two pieces of information: an explicit

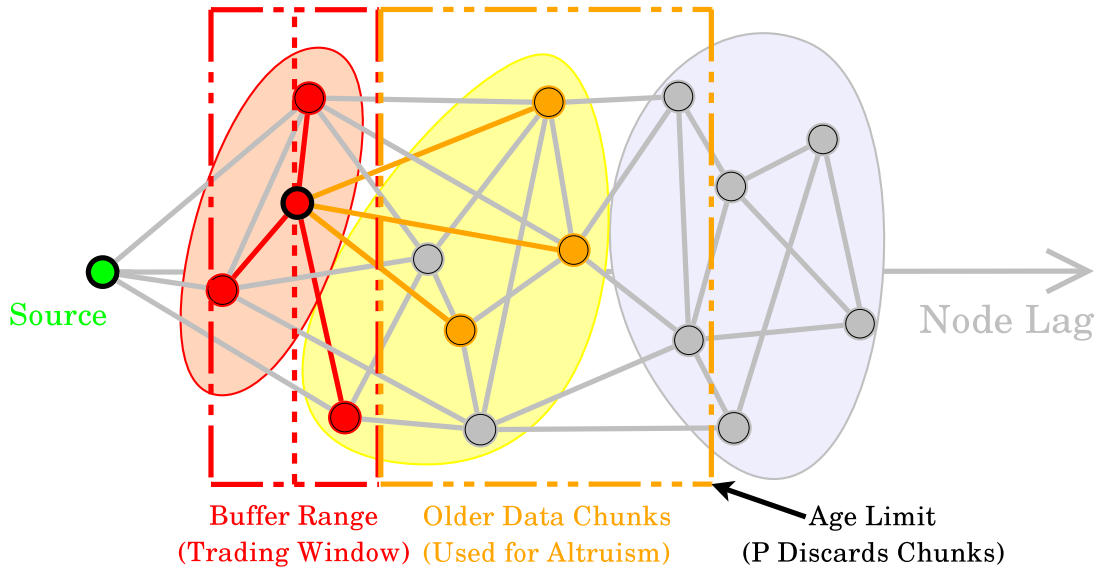


Figure 4.2 : A PULSE peer and its exchange sets (MISSING and FORWARD)

one, that is the range of chunks a node is able to serve, and an implicit one, which is an estimate of the peer's trading capabilities related to the incentive mechanisms. Intuitively, cooperating peers will be able to receive new chunks faster than selfish ones, and thus will find themselves nearer to the source, i.e. have lower $T_{B_{avg}}$ values. Node decisions are based on the currently available local knowledge, which includes:

- information about the *address* and buffer delay range $[T_{B_{avg}}, T_D]$ of the other peers. This information is forwarded among peers by means of low-priority messages, called *BLUE* messages, using a gossip/epidemic protocol such as SCAMP [43]. Nodes known with this level of detail are inserted in the *BLUE knowledge list* and can be selected as candidates for the exchange of more detailed information about the buffer status. The set of BLUE peers assures connectivity between different nodes of the system and represents the basic level of knowledge.

- detailed accounts of the exact content of remote peers' buffers such as the instantaneous node position $T_{B_{inst}}, T_D$, a bitmap summarizing the chunks present in the trading window, and (optionally) explicit request bitmaps for chunks in that range. This information is propagated by means of pairwise exchange of local messages, called *RED* messages. Every node periodically selects a subset of peers in the BLUE list and sends them such information. A node receiving a RED message responds with the same information. RED messages are also exchanged with peers involved in chunk exchanges. Nodes known with this level of detail are inserted in the *RED knowledge list* and can be selected as targets for chunk exchange.

- direct measurements of network parameters: RTT estimated during the exchange of RED messages, and *data throughput per connection* measured during chunk exchange.

- local records of previous trading interactions, in the form of a cumulative history score H.

Considering the volatility of the information exchanged and the network dynamics, the knowledge is not persistent but valid only for a limited period of time. Indeed, knowledge lists are managed according to timeouts: a peer in the RED list is moved to the BLUE list after a timeout, and a peer is removed from the BLUE list after a longer timeout. Anyway, all nodes that have been contacted during the peer lifetime are stored in a list and recontacted to obtain up-to-date information if needed.

4.1.4 Resource allocation algorithms

In a mesh-based live streaming system, node resources are exploited in order to exchange chunks between peers. In particular, the most critical resource is the network bandwidth so that diffusion algorithms are in charge of managing the network capacity of nodes. These algorithms are driven by peer/chunk selection policies and take decisions on the basis of local knowledge.

In PULSE, the selection is performed in the *peer-then-chunk* order. Periodically, the local peer selects a set of nodes among the peers for which it has RED knowledge. Chunks are then selected for each of them according to the buffer status. The algorithm to select the peers to which sent RED messages is therefore crucial for chunk dissemination as well: it should be designed according to the peer selection strategy in order to provide it a good set of peers over which perform the selection.

We first describe the chunk/peer selection policies; these algorithms take as input the status of the local peer buffer and the information available in the RED list. After that, we are going to describe the algorithm used for the selection of RED message targets.

Peer selection

Each peer periodically performs peer selection. Its time period is called *EPOCH* and is of constant length T_e. The two peer selection algorithms used in PULSE are: an optimistic tit-for-tat selection based on the total bandwidth received during the previous EPOCH (similar to the one used in BitTorrent), and a lag-constrained selection based on a cumulative trust score. The two algorithms are executed at the beginning of every EPOCH, and give as a result two lists of peers, the MISSING and the FORWARD lists. The local node will attempt to associate and exchange data with these peers throughout the next EPOCH. A third list of nodes, called *new* list, is filled with peers that are sending data during the current epoch and are not included in the other two lists. The role of this list is to immediately reciprocate nodes that are sending data to the local peer in order to establish a pairwise exchange.

- **Optimistic Tit-for-Tat**

  This policy aims to identify which peers, among all those about which a node has RED knowledge, are currently interested and able to provide data in the short term. Two pieces of information are then relevant to this choice: the fact that a peer has provided data in the recent past and may be expecting a short-term compensation to continue to do so, and the presence of a shared interest in the same window of the stream which may lead to fruitful future exchanges. The selection policy we employ in PULSE uses a tit-for-tat choice based on information about the amount of data received during the previous

EPOCH to fill the m_{miss} upload connections of the MISSING list. At least one place in the list is reserved for an optimistic selection, leading to the choice of a known node with the largest trading window overlap (network latency can be taken into account to bias this selection toward peers in the vicinity).

In details, all the peers that sent data to the local node during the last EPOCH are ordered by the number of non-duplicate chunks it received from them. A configurable quota of $m_{miss} - m_{BI}$ connections is then established with the highest-ranked nodes. The remaining m_{BI} ($m_{BI} \geq 1$) connections are allocated according to a *latency-biased interest function* BI. The latency-biased function between a peer l and a peer v is defined as: $BI(l, v) = I(l, v) - W_{RTT} \cdot \frac{RTT(l,v)}{2}$, where the W_{RTT} parameter is called the *latency weight* and $I(l, v) = W_S - |T_B(l) - T_B(v)| = W_S - \Delta T_B$. I estimates the reciprocal interest on the bases of the average buffer reception delay T_B while BI is introduced to take into account network latency between peers and enforce locality awareness in chunk exchange.

- **Altruistic-Historic based**

  Every node maintains a record of the previous interactions with every other peer as a numeric value, which we refer to as the *history score*. This mechanism enables a peer to build a knowledge base about its fellow peers: its goal is to gather data on past behavior that will allow nodes to make better *informed choices* when selecting future candidates for FORWARD exchanges.

  The score is computed as follows: each time a previously unknown peer is encountered, it is given an initial positive score. The score is incremented by a fixed value whenever some useful chunks are received from a node while it is not present in any of the local exchange lists. The node's score is decreased by some fixed quantity whenever it is chosen as FORWARD partner and receives one or more chunks from the local peer during that EPOCH.

  In details, peers are ordered by decreasing history score, and selected only if their trading window is not overlapping with the local trading window (i.e., the remote node is currently "farther" from the source than the local peer). Nodes already belonging to the MISSING list are ignored.

At any given moment, each peer must maintain several connections for sending and receiving data. To simplify the problem of bandwidth allocation, PULSE peers try to establish a fixed number of outbound connections for data exchange, but do not limit the number of incoming ones.

A small number of connections m_{miss} (e.g. four) should be reserved for MISSING partners: this number should be chosen so that a peer with a barely sufficient upload rate (near SR) can attain a reasonable theoretical throughput on each connection (e.g. SR/4). Increasing the number of MISSING connections will increase the control and computational overhead, while reducing the effectiveness of the tit-for-tat selection. We must in fact remember that at steady state, under a *rate-limited* application such as live streaming, no more than SR will be received in average by a node. Intuitively, raising the base number of connections means lowering the expected throughput from each MISSING partner. This will in turn increase the impact of "noise" on the tit-for-tat peer selection, and eventually undermine the overall system stability. On the other

hand, especially for the richer nodes, opening more connections could improve the odds of finding useful chunks and fully exploiting their capacity. To take this fact into account, a variable number of connections m_{forw} can then be assigned to FORWARD exchanges, depending on the available outgoing bandwidth. These connections will allow resourceful peers to donate their excess bandwidth to the system by providing other peers with chunks with no expectations for an immediate return.

If some peers that are not included in the MISSING and FORWARD sets send data to the local peer, this last adds them to the NEW list. If there is extra bandwidth with respect to the amount needed for MISSING connections, peers in the new list are served with priority with respect to FORWARD connections in order to develop a continuous relation of chunk exchange.

A detailed analysis of the impact of the number of outgoing connections per peer set is available in [143].

- **About the source**

  The source differs from the other nodes since it doesn't need to engage in exchanges to get data chunks. It always has a complete sliding window, and its lag value is zero by definition. As a consequence, the peer selection algorithm at the source also needs to be different.

  Moreover, the source lacks the data exchange feedback mechanism, and could be exploited by malicious nodes that try to retrieve all chunks directly. The attackers could then avoid contributing to the system and may even put in danger the entire distribution process if the source's upload bandwidth is small. To mitigate this danger, the source has to change the subset of nodes it serves at each EPOCH, and must not send groups of contiguous chunks to the same peer.

  The peer selection algorithm we employ is similar to the one used by seeds in the latest BitTorrent software versions [60]. At the beginning of each EPOCH, the source prepares a list of known peers that have a T_B value (instantaneous or average) smaller than a fixed threshold. It then chooses randomly a small subset to which it will send chunks during this EPOCH. The source treats this list as its MISSING list. The source has no FORWARD or NEW list because, as explained before, it doesn't download any data from the other nodes.

Chunk selection

As stated in 3.1 chunk selection cannot be completely push or pull in real system but based on an hybrid approach where every chunk exchange is the result of a *negotiation* between provider and recipient peer.

PULSE is therefore based on an hybrid push/pull chunk exchange mechanism. A receiver node requests chunks similarly to the heuristic used in DONet/CoolStreaming [130] or in BitTorrent [30]. Its purpose is to request the *rarest chunks* among those that are locally available, and to distribute the requests across different possible providers. Using the local knowledge gathered from the current RED set, chunks that are rarest across the neighborhood are requested with higher priority than more common ones. Chunks with an equal number of providers are preferentially requested to neighbors that recently provided data to the local node, because they

will likely provide them. To limit the load on any single peer, the maximum number of per-node requests is bounded. A chunk is awaited for a certain period of time after which, if it has not been received, it is requested from another peer

At sender side, the chunks to be sent over a connection, regardless if MISSING or FORWARD, are selected by using a *Least Sent First / Random* strategy. Each peer keeps a counter of how many times it has sent each requested chunk. The one that has been sent the least number of times is chosen to be sent first among the requested ones. In case of a tie, the chunk is selected randomly.

Selection of RED message targets

The peer selection is performed among nodes for which the local peer has a RED knowledge. Every node periodically selects some peers from the BLUE list and exchange RED messages with them in order to spread/retrieve this level of information to/from its neighbors.

These target nodes should be selected in order to provide to the peer selection algorithm a representative and useful set of potential recipients. Since the peer selection policy selects both peers with overlapping and non-overlapping trading window, the set of nodes to which the local peer sends RED messages should contains nodes belonging both categories. These peers are periodically changed and RED messages are sent to them at a constant rate.

Additional RED messages should be sent at higher rate to peers that are providing data to the local node and to peers that are downloading data from the local node. In fact, fresher information about the buffer status of such peer is required in order to find useful chunks to download/upload. Please refer to [146] for more details about the policy used for the selection of target peers of RED messages.

4.1.5 Implementation details

We developed a PULSE prototype that is written in Python and uses Twisted, an event-driven networking framework. The prototype also uses an external library for the Reed-Solomon FEC [99], that is applied to chunks as fixed-rate error correction code.

A PULSE node can be whether a source or a normal peer, the type being specified as parameter at launch. In case the node is a source, the prototype takes as input from a source the stream (e.g. a webcam or a video file), and generates a *.pulse*. This file has same role of a *.torrent* file in a BitTorrent session and must contain:

- A set of one or more addresses of peers that are part of the streaming session when the file is generated. The joining peer contacts one of these entry points by generating a BLUE message that is then ged by the contacted node.
- Information about the parameters of the stream and of the PULSE protocol. These parameters are used to set the application parameters and to correctly play out the stream.

Additional information such as metadata of the stream (title, author and so on), security mechanisms like source public key or signature, can be added to this file. Differently from a *.torrent*

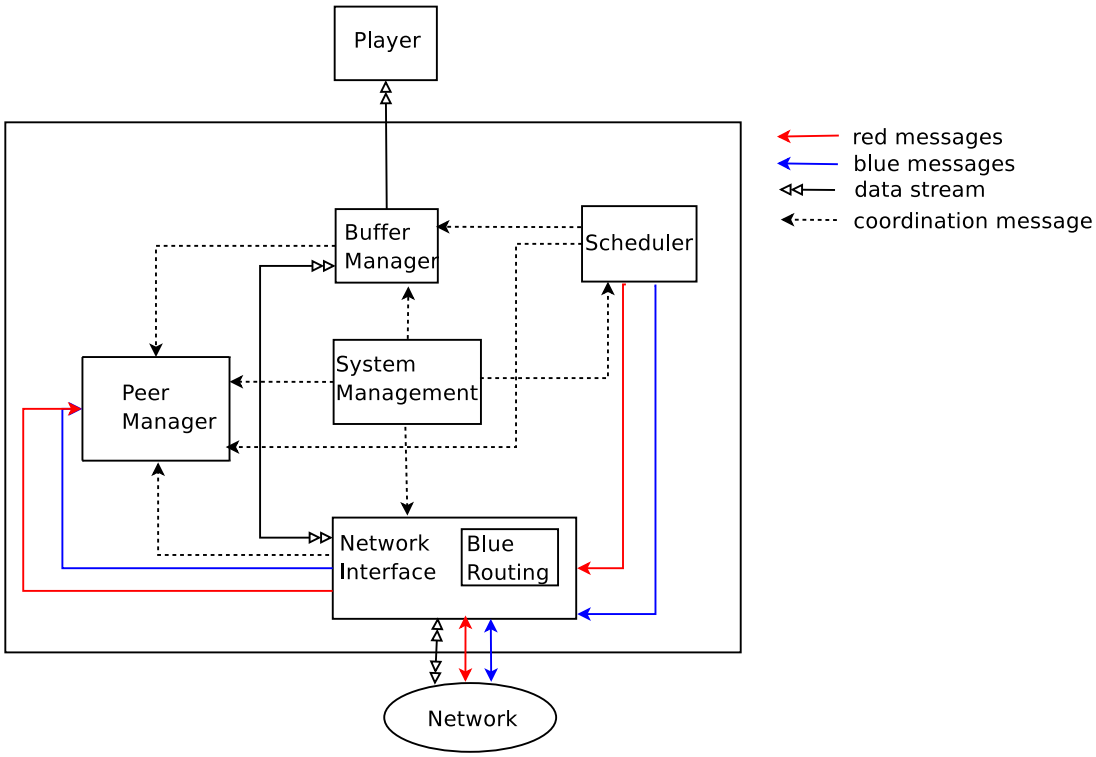


Figure 4.3 : PULSE prototype structure

file, the *.pulse* file does not contain the hash of the chunks because they are generated on the fly and therefore not known when the file is generated.

In case the node is a common peer, the prototype uses the *.pulse* for the start-up phase, and when running, it outputs the stream ready for the play out.

The general structure of the prototype is presented in Figure 4.3. The *Blue routing* module contains the implementation of SCAMP [43] that is used for the forwarding of BLUE messages. The *buffer manager* is responsible for the management of the received chunks, for the different buffer parameters and for the chunk coding/decoding. The *peer manager* module is responsible for the management of the knowledge lists and for the peer selection process. The *scheduler* module is responsible for chunk selection and sending, and for the exchange of RED messages. Finally the *system* management is responsible for the coordination and the execution of the different algorithms.

In order to compute the lag of received chunks and the different system parameters, nodes should be synchronized among them. This is achieved in PULSE by using a simple synchronization mechanism that computes the offset between a given node's clock and the source's clock.

As transport protocols, PULSE uses TCP for chunk exchange and UDP for control message forwarding (BLUE and RED). The choice of TCP is motivated by the need of a reliable transport protocol in order to assure the correct transfer of a chunk once scheduled by the application. Considering the time constraints of live streaming, chunk sizes should be smaller than in a file-sharing context and may range from few to several tens of kilobytes. This may cause rate slowdown due to the congestion window that would not have the time to grow before the upload of a chunk ends, and may make the TCP overhead significant with respect to the data transfer.

To improve TCP efficiency, in the prototype we try to pipeline chunks directed to the same neighbor. In particular we decouple the chunk scheduling from the chunk transmission by using a per-neighbor sending queue. This queue is filled of few chunks by the chunk scheduling algorithm, and a new chunk is scheduled only when another leaves the queue because its upload has been completed. In this way the local peer continuously sends data to the considered neighbor and avoid to over-load peers with low download capacities.

To use UDP would solve the TCP concerns and would eventually permit lower chunk sizes leading to shorter chunk upload times and thus lower diffusion delays. However, a recovery mechanism at transport layer, like FEC, would be needed. In fact, without a reliable transport protocol the whole chunk is lost if even only one UDP packet is lost. Another solution would be to explicitly notify the reception of a chunk. This would avoid extra-coding at transport layer but, as before if only one UDP packet is lost, all the bandwidth used for the chunk transfer is wasted because the chunk should be completely retransmitted.

A second version of the PULSE prototype implements UDP for data transfer with a FEC coding applied over UDP packets of a chunk. We are currently running some experiments in order to evaluate the impact of using UDP as transport protocol for both access and core constrained networks.

As concern control messages, we do not need a reliable transport protocol because the loss of RED or BLUE messages may be tolerated by the application. Moreover, such messages are typically very small (about 100 bytes) making TCP unsuitable for their transport.

PULSE is currently developed by the Napa-Wine project [79] and the P2PIm@ge [84] projects that provide new releases of the code.

4.2 Related work

The first working prototype of a live streaming application dates back to 2002 when Peercast [34] has been released and an experimental evaluation has been performed over a small scale testbed. EMS [25] is the first large-scale video distribution system based on a single-tree infrastructure. Performance evaluation has been performed by collecting logs of users; this experience shows overlay multicast is feasible, and highlights issues and limitations of a single-tree architecture.

In 2003 a prototype of Splitstream [18] has been developed and tested over a small number of peers in PlanetLab. This experience shows the feasibility of a multiple-tree approach for live streaming systems. An experimental evaluation of Bullet [57] shows an hybrid mesh-tree approach can achieve higher throughput than a simple tree-based system. Chainsaw [86] is the first to experimentally show a fully mesh system can achieve the same performance of Bullet and Splitstream while recovering faster from node churn.

More recent works focus on rate/delay performance analysis of distribution algorithms. An evaluation performed over PlanetLab and presented in [47, 62], shows that an algorithm spreading chunks over spanning trees of limited depth can achieve high diffusion rates. [102] shows, by means of network emulation, that incentives in live streaming can improve the performance of a tree-based system. As concern mesh-based application, in [129] the achievable diffusion rates and delays of an hybrid push-pull algorithm are analyzed over PlanetLab and in a real

deployment over Internet. Picconi and Massoulié [90] show there exists algorithms that can achieve close to optimal diffusion rates in real conditions by means of network emulation.

In last years, several commercial live streaming applications have been released and have become increasingly popular. As a consequence, lot of attention has been devoted to the experimental evaluation of systems like CoolStreaming [130], PPLive [95], UUSee [53], TVAnts [114], PPStream [96], and SopCast [105]. First evaluations have been proposed in [5, 104], where such systems are studied by means of packet level analysis. [5] considers PPLive and Sopcast and focuses on time evolution of metrics like transmitted/received bytes, number of provider peers and so on. [104] considers PPLive, PPStream and SopCast, and proposes a flow level analysis. Large-scale measurements performed in the context of the Napa-Wine European project [29, 28] analyze the performance and the network awareness of PPLive, TVAnts, and SopCast by means of packet level analysis.

Crawling techniques have been used to analyze the PPLive protocol [49], the QoS it can provide to users [50], the importance of stable peers on system performance [117], and the overlay characteristics [116]. Logs collected at both server and peer side, have been analyzed to understand the behavior of the old and the new CoolStreaming [130, 61], and the impact of the number of channels on server load in UUSee [121].

This analysis of commercial systems highlight they are all based on a mesh approach. First evaluations observe some applications use mainly TCP for data transfer and mainly UDP for control exchange, like PPLive, TVAnts and PPStream, while others use mainly UDP for both, like SopCast, ([104]). More recent analysis shows that the current trend is to use UDP also for data transfer ([29]). Awareness in peer selection focuses on the upload capacity of nodes, while only PPLive and TVAnts take into account Autonomous Systems nodes belong to. However, higher locality awareness and incentive mechanisms have not been observed ([28]).

Such studies also highlight users behave like common TV watchers and the applications achieve quite high start-up and playout delays, ranging from several seconds to minutes ([49]). It also turns out that, despite the high number of users, the main contribution to the system is provided by some stable and well provisioned peers ([117]), and a quite high number of servers is needed to increase the number of broadcasted channels ([121]).

Recently, also multiple-tree systems have been deployed over the internet like Stanford Peer-to-Peer Multicast (SPPM) [82] and Peerialism [88]. A large-scale evaluation of the former [81] highlights such system can achieve delays of few seconds with a miss ratio of about 1%. However, a fair comparison is not possible because they are still much less popular than commercial mesh-based systems (e.g. 1000 simultaneous users for one channel for SPPM versus the 100000 of PPLive).

[4] also considers tree-based commercial systems and compares them to mesh-based ones[2] on a small scale test-bed. Main results highlight tree-based systems require less overhead, while mesh-based ones provide lower start-up delays, are more network aware and more adapted to network dynamics.

In this chapter, we consider PULSE as a mesh-incentive approach to live streaming and we verify whether it meets the application requirements or not. In details, we analyze the rate/delay performance, and we study the data path characteristics in order to better understand the diffusion process in such kind of systems.

[2] Analyzed systems are anonymized.

Class	Upload bandwidth	HH-LB	LH-LB	HO-LB
Very Rich (VR)	4 SR	4%	0%	0%
Rich (R)	2 SR	20%	20%	0%
Normal (N)	SR	21%	80%	0%
Poor (P)	SR/2	55%	0%	0%
Normal+ (N+)	1.1 SR	0%	0%	100%

Table 4.1 : Bandwidth scenarios for Grid5000 experiments

4.3 Performance evaluation

We perform a first set of experiments on *Grid5000* [93], an highly reconfigurable, controllable and monitorable experimental platform distributed over nine sites in France and one in Brazil. It has been developed by the Grid5000 project that aimed to provide 5000 CPUs for grid computing and network application testing. The project goal has been reframed at 5000 cores, that was reached during winter 2008-2009. The french sites are interconnected by the Renater Education and Research network that provides links at 10 Gbit/s.

Grid5000 is a very useful instrument for testing the PULSE system because everything is under the control of the user and no external factors can alter the results. A user reserves, for a certain period of time, a certain number of machines where it is the only person allowed to run tasks. By doing this, the user can be sure that results won't be affected by high CPU load. The high speed links assure negligible and constant latencies between the different nodes, and circumvent bandwidth bottlenecks.

On this platform we perform tests by artificially limiting the peers upload capacity in order to emulate the effects of upload bottlenecks at access links. This is possible by implementing a configurable upload bandwidth cap into the prototype software. Unless otherwise stated we suppose all nodes join the system at the beginning of the experiment: for technical constraints this joining phase lasts about 20 seconds. Considering the negligible latencies between nodes it is not possible to evaluate the latency awareness of PULSE, so we set the latency awareness parameter to $W_{RTT} = 0$. The bandwidth scenarios we use for our experiments are reported in Table 4.1 while the system and stream parameters are reported in Table 4.2.

The *High Heterogeneity - Low Bandwidth* scenario (HH-LB) corresponds to a very pessimistic bandwidth distribution: not only the upload capacities are heavily asymmetric, but more than half of the nodes can contribute no more than one half of the original stream rate. The percentages are loosely inspired by the results of the study by Sariou et al. on Gnutella peers [101] that showed an approximative power-law distribution of upload capacities higher than 10 Kbps. In such scenario the maximum achievable download rate per peer is only 4% higher than the stream rate (see Chapter 2): this means there is just 4% of extra bandwidth with respect to the capacity required for the feasibility of the system.

The *Low Heterogeneity - Low Bandwidth* scenario (LH-LB) portrays a system where there is an extra bandwidth of 20%, that is evenly split among a minority of the population. The challenge in this scenario is given by the small difference of capacity between the two classes, and the relative scarcity of bandwidth.

Parameter	Value	Description
n	1000	Number of peers
u_S	4 SR	Source upload capacity
W_S	32	Sliding window size [chunks]
$TW = 2\,W_S$	64	Trading Window size [chunks]
W_{RTT}	0	latency weight
T_D	$T_B + 12$	Time at which a chunk is discarded from the buffer [seconds]
T_e	2	Epoch length [seconds]
m_{miss}	4	Size of the MISSING set
m_{BI}	1	Missing slot reserved for optimistic unchoking
m_{forw}	8	Size of the FORWARD set
SR	256	Stream rate [Kbit/s]
λ	8	Chunk generation rate [chunks/s]
c	4	Chunk Size [KB]
$miss\ ratio_{max}$	20	Maximum percentage of missing chunks

Table 4.2 : System and stream parameter for Grid5000 experiments

The *Homogeneous - Low Bandwidth* scenario (HO-LB) aims to reproduce a system where all peers have the same upload capacity and there is just 10% of extra bandwidth. This scenario is mainly useful for an easier and deeper understanding of the properties of the chunk distribution mesh.

Metrics

To evaluate the performance of PULSE we should adapt our metrics in order to take into account the system functioning and structure.

As an indicator of the *diffusion delay* we are going to monitor the evolution of $T_{B_{avg}}$ (simply named *average lag* in the following) of a node over time. In fact, this parameter represents the average age of the chunks in the trading window and varies according to time the node takes to retrieve chunks.

As concern *chunk miss ratio*, PULSE limits this value to the $miss\ ratio_{max}$; if not enough chunks are retrieved to fit this requirement the sliding window stagnates and the buffer eventually disconnects. We are therefore going to monitor both the chunk miss ratio and the number of buffer disconnections suffered by peers in order to understand the diffusion rate performance of PULSE algorithms.

Finally we analyze the properties of the chunk distribution trees: in particular we focus on tree depth and width.

All these metrics are going to be investigated on a per-class basis in order to analyze the resource awareness achieved by allocation algorithms.

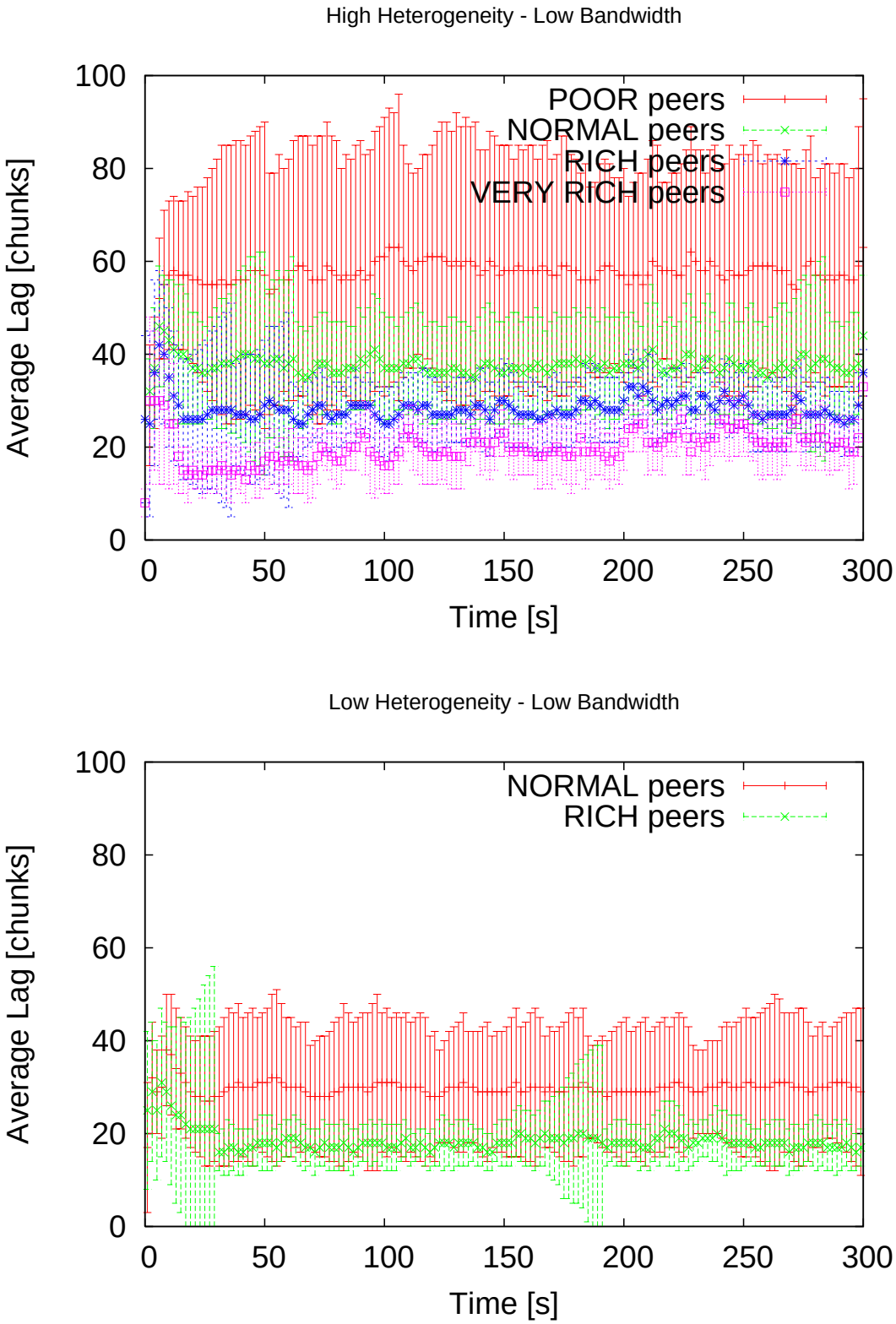


Figure 4.4 : Per class average lag evolution over time in Grid5000 experiments. Lag variance is also reported.

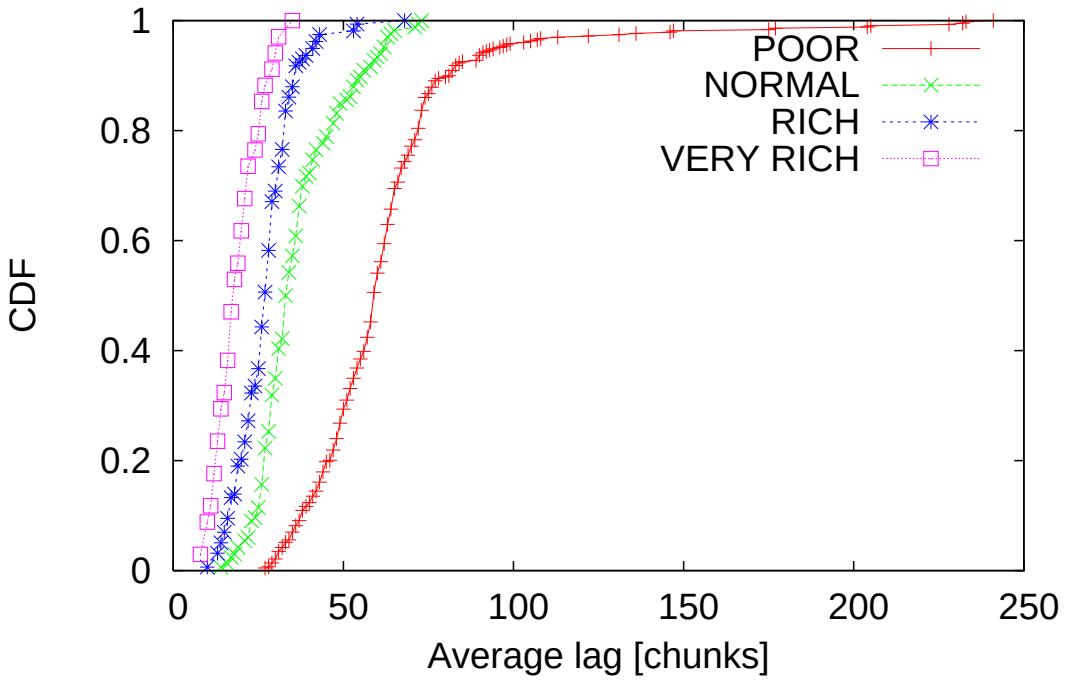


Figure 4.5 : CDF of average lag over peers at 150 s for the HH-LB scenario.

4.3.1 Average lag and chunk miss ratio

In Figure 4.4, we plot the per-class average over time of a node's T_B for a typical run, under HH-LB and LH-LB scenarios respectively. We immediately see that the average T_B for each class is very stable over time in both traces, with minimal fluctuations. The biggest event can be observed between $t = 160\ s$ and $t = 190\ s$ in the LH-LB run, where the standard deviation of the RICH class increases and then falls back to normal values. The visual appearance of the fluctuation is indeed much greater than its relevance, as what happened is that eight RICH nodes had their lag slightly increased for a short time, with two of them once reaching a T_B of 80 chunks, but without any consequences in terms of data loss or playback disruption.

The most striking result conveyed by Figure 4.4 is the strong relationship between the available upload bandwidth of a class and the average lag of its members: peers with the highest bandwidth contribution reach in both cases a steady-state lag of about 20 chunks (that is, less than 3s) from the media source. On the other hand, the less a class contributes, the worse its average lag: the POOR class in HH-LB gets the highest average lag among the four, at nearly 60 chunks (more than 7s). Visually, the plot for HH-LB is especially telling, as the four classes appear *sorted by resources and layered one after the other*, with a meaningful difference between the average lag performance of each class: also, we notice that *the lag difference becomes higher when a class' available upload capacity is smaller than the stream bandwidth*. The same remarks can be made about the LH-LB scenario, as the lag difference of the two classes is smaller but still evident (18 chunks or 2.2 sec for RICH, 30 chunks or about 4 sec for NORMAL).

Figure 4.4 also depicts the lag variance as vertical lines. Interestingly, the variance for all classes also quickly converges to a stable value, which is again related to the upload contribution: the variance of T_B increases as the upload bandwidth available to each class decreases. This suggests that the fact of having more bandwidth not only reduces the average lag, but also tends to give nodes a more stable position in the system. This can be seen both in the HH-LB and

in the LH-LB scenarios: the correlation between upload capacity and lag stability is evident, resulting in a much lower standard deviation of lag performance for the resourceful classes (4-10 chunks for VERY RICH and RICH, vs. 10-25 chunks for NORMAL and POOR in HH-LB; 4 chunks for RICH vs. 11 chunks for NORMAL in LH-LB). This suggests that *having more bandwidth not only reduces the average lag, but also tends to give nodes a more stable performance in the system*. This is confirmed by Figure 4.5 that reports the CDF of the average lag perceived by peers per class. Again the more a class contributes to the system, the lower is its delay and the more the delay distribution is centered around the average value.

The two previous observations are important, as they represent a kind of incentive that can appeal to rational agents. It is indeed in the best interest of any node to provide at least as much bandwidth as it demands, as the typical consequences of doing so result in a better performance of nodes that provide sufficient resources. Providing less than that is allowed, but higher lag and temporary disconnections should be expected.

The analysis of upload and download bandwidth utilization (Figure 4.6) also confirms the system's stability: all classes contribute an average total bandwidth which is almost constant over time, and receive chunks at rates that are always sufficient to reconstruct the original media stream from its FEC encoding. In details, in the HH-LB scenario all classes have a chunk miss ratio of about 15% while in the LH-LB the miss ratio is of about 5% because of the larger bandwidth over provisioning. Since $miss\ ratio_{max} = 20\%$ in both scenarios, peers can recover these losses thanks to the FEC mechanism.

Considering the way data exchange algorithms work, the progression of a node buffer can be delayed for some time because of missing chunks in its trading window - in fact, this happens routinely - but bandwidth usage data confirm that the global impact of these short starvation periods is low, and recovery is fast. As a matter of fact, no node ever experienced disconnections during all of the tests we performed.

4.3.2 Data exchange

In this section we examine the effects and consequences of peer selection on the global scale of the system, as it is captured by our measurements. To better understand the actual relevance of the various selection mechanisms, we collect quantitative data about the chunk exchange process and correlate it with the peer selection outcomes. We keep track, for each chunk, of the class to which the sender and receiver belong, and of the type of the connection (whether it's MISSING, FORWARD, or NEW) as determined by the sender peer. In Figure 4.7 we display the data we collected over a typical HH-LB run, as it is seen both from the uploader's and the downloader's point of view.

The first finding is that MISSING exchanges alone do convey in average more or less the stream bandwidth. Even when peers may be able to provide much more, the bandwidth is not used, and would probably be wasted to a large extent if no FORWARD connections were present. Another remark is that cumulative outcomes of MISSING exchanges do roughly match the affinity rating between each class pair. This indicates that MISSING connections provide in average a steady flow of data to and from neighbors with overlapping trading windows.

A second result is that FORWARD connections are especially important to distribute the excess capacity of the richest classes to the poorer ones: FORWARD exchanges from VERY RICH and RICH peers alone to nodes in the POOR class provide, in average, almost half of the POOR

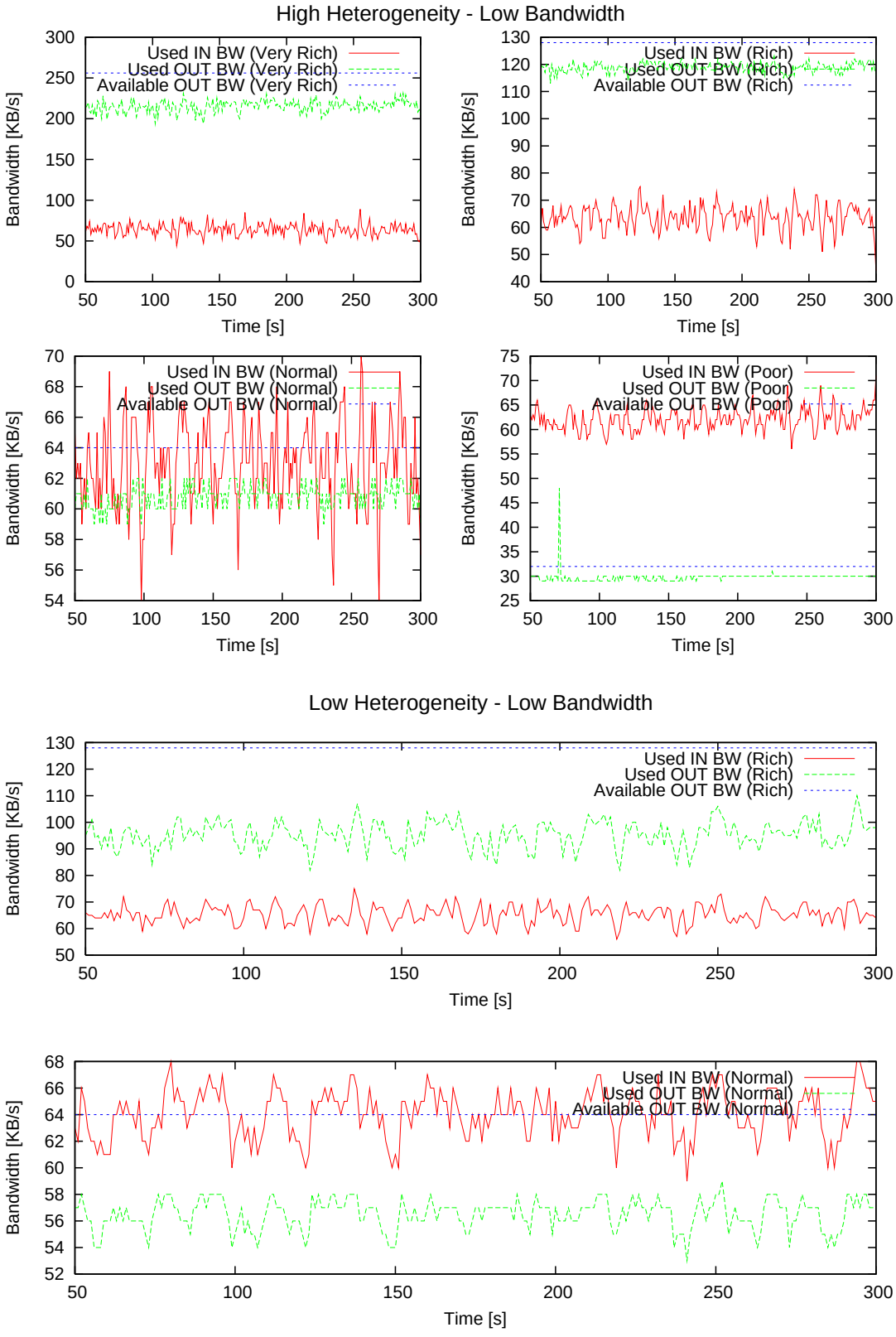


Figure 4.6 : Download and upload bandwidth utilization.

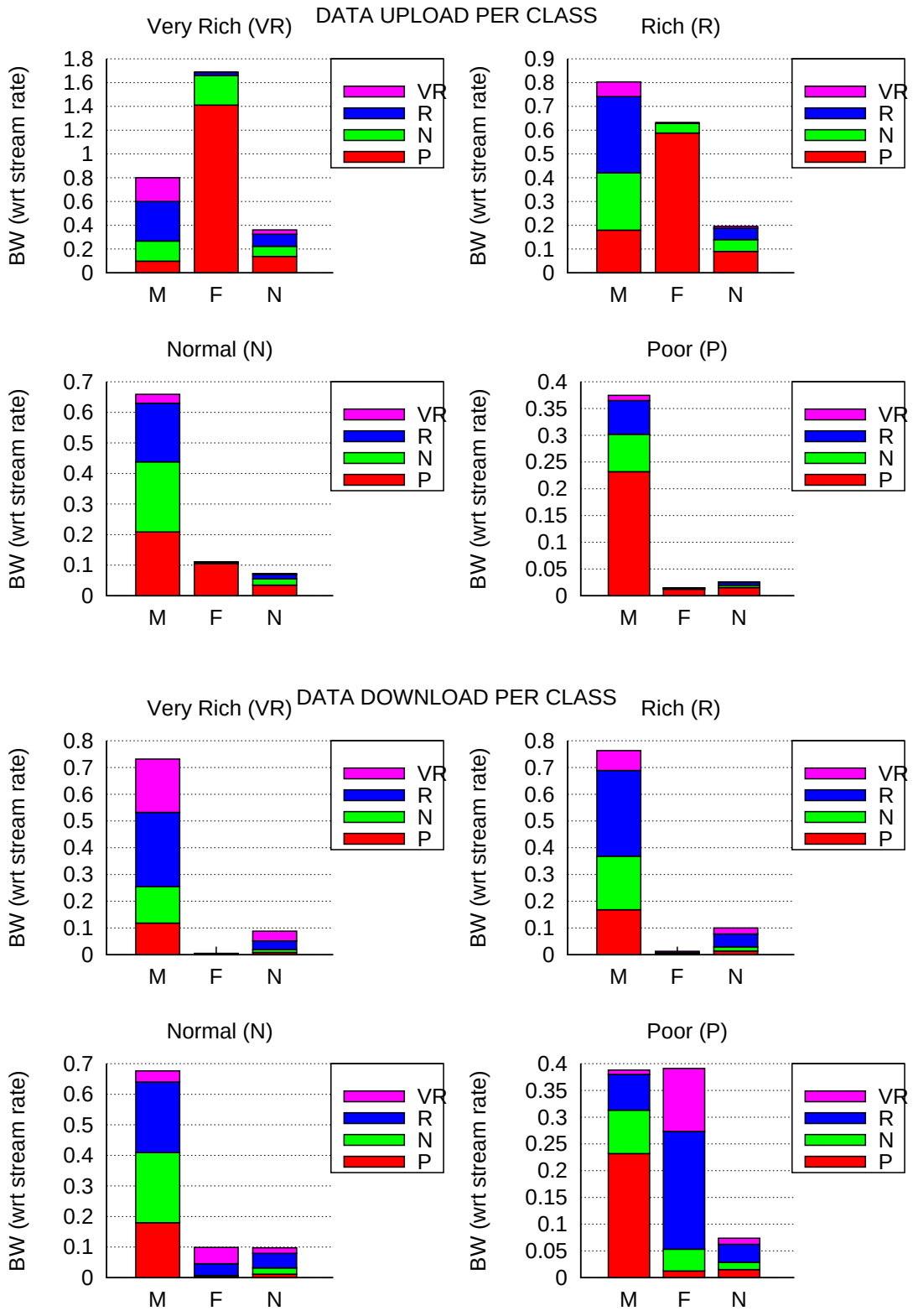


Figure 4.7 : Data exchange between classes for the HH-LB scenario.

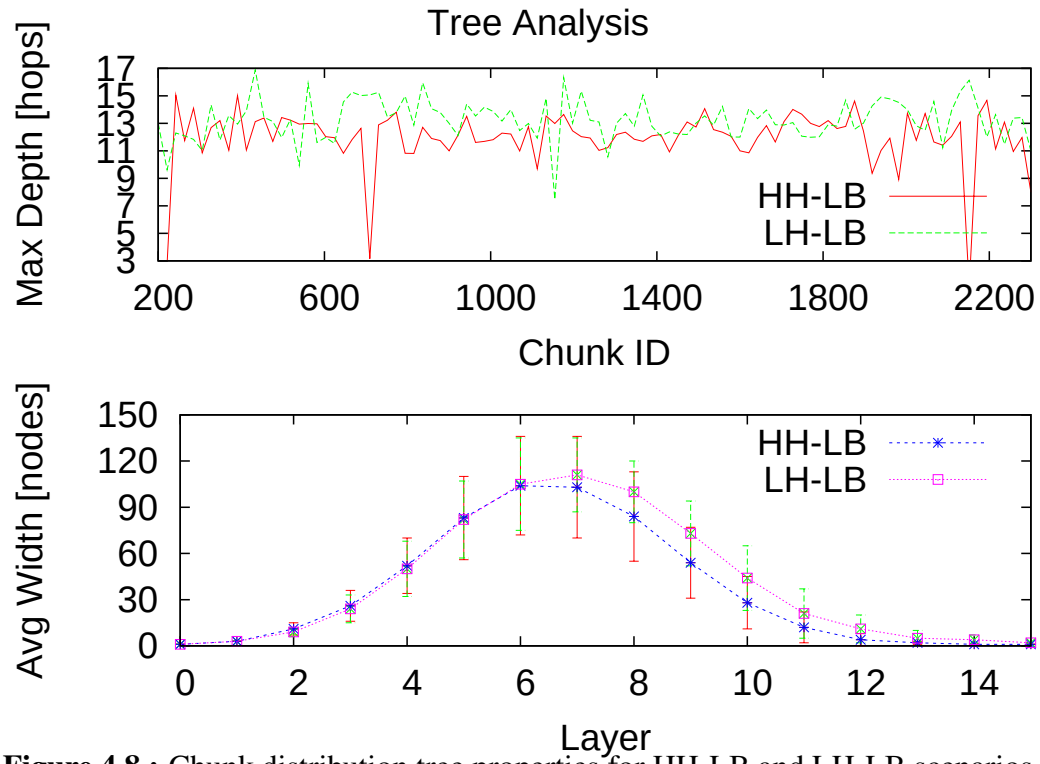


Figure 4.8 : Chunk distribution tree properties for HH-LB and LH-LB scenarios.

stream rate, while they obtain the rest mostly from MISSING exchanges with other POOR peers. On the other hand, the scarce bandwidth of the poorest classes (up to the stream rate) is rarely allocated to FORWARD or NEW connections.

As expected, data obtained through NEW connections represents a small fraction (about 10%) of the stream rate. In fact, the NEW exchange set is designed to fasten the establishment of a stable relation between nodes and not to provide large amount of data.

These results confirm that the PULSE algorithms are correctly exploiting the available capacity: tit-for-tat based MISSING exchanges are important under bandwidth scarcity to ensure a proportional exchange reciprocation, whereas FORWARD exchanges based on lag and node history allow to distribute the unused resources evenly to the entire system. This highlights that, in a chunk diffusion algorithm,3 part of the upload bandwidth should be devoted to incentive or resource aware exchanges, while the remaining should be altruistically distributed. Both mechanisms, the resource aware and the altruistic selection, should be employed and play a critical role for the system stability and performance. A detailed analysis of the awareness-agnostic trade-off in allocation algorithms is presented in the next Chapter.

4.3.3 Data path analysis

We have seen above that the position of the nodes in the incentive-generated data exchange mesh is related to their bandwidth contribution: we are now interested in analyzing what is the specific impact of this global node placement on the distribution process of individual data chunks. To this end, we study the paths taken by data chunks as they are replicated by the nodes.

Layer	1	2	3	4	5	6	7	8
% VR	3	20	21	15	7	2	1	0
% R	15	35	40	41	34	18	6	2
% N	19	21	20	20	24	27	18	8
% P	62	24	19	24	35	53	76	90

Table 4.3 : Average observed composition of distribution tree layers by bandwidth class (HH-LB)

As no duplicate chunks are allowed, the resulting directed distribution graphs for each chunk are free of cycles (i.e. single trees). The average properties of these trees (width, depth) can provide precious insights to complete our observations on node lag.

We show the analysis of the average properties of chunk distribution trees in Figure 4.8. We notice that the maximal tree depth[3] in hops for individual chunks is short and quite stable over time. In our system with 1000 nodes, maximum tree depths are in average between 11 and 14 hops, for both bandwidth scenarios. Without any explicit structural guidance, the paths taken by the chunks are consistently good, even under a widespread bandwidth scarcity and while the data connections between nodes are continuously renegotiated. More information can be gathered by the statistics on tree width: from Figure 4.8 we can appreciate the fact that the first few layers of the trees are in average very wide, and that the average percentage of nodes that find themselves placed in the last few layers of the trees is low ($<$20%). Also, the variance of layer width is high for the first few layers, with a standard deviation in the order of up to 30% of the average value.

Several details can be also seen in Figure 4.8, if we look more closely: we can notice how HH-LB trees are in average a little *shorter* than LH-LB trees, despite the fact that the resources available in the HH-LB scenario are less than in the LH-LB. Average tree widths are also very similar, especially in the first few layers, where HH-LB tops LH-LB by a small margin. These observations can be explained if we take into account the effects of the incentive-based peer selection mechanism: remember the source randomly selects its targets for data exchange among the nodes with lowest lag; we then observed node clustering by upload capacity, with the richest classes constituting a large fraction of the peers with low lag values. The net effect of these two combined mechanisms is that the chunk distribution trees from scenarios with high levels of heterogeneity can have very wide initial layers, due to the richest peers being on top. Wide initial layers are very important in the context of live media distribution, since they reduce considerably the maximum number of hops a chunk has to traverse to reach all the nodes. To verify our conjecture, we analyze in Table 4.3 the average placement of nodes that belong to each bandwidth class in the chunk distribution trees. We can observe that there is indeed a preponderant presence of peers from the richest classes in the few first layers, especially between layer two and five, where roughly half of the peers have the necessary resources to replicate each chunk more than once and up to four times.

For an easier understanding of the properties of chunk distribution trees of PULSE we now consider the homogeneous scenario. This scenario also allows us to compare PULSE tree properties to structured systems, that have mainly been designed for homogeneous scenarios. We

[3]Chunk distribution trees obviously only include nodes that receive a certain data chunk, thus the number of nodes in a tree can change on a chunk-by-chunk basis. However, the protocol mechanisms guarantee that connected nodes can never lose in percentage more than the the FEC rate (in our case, 20%).

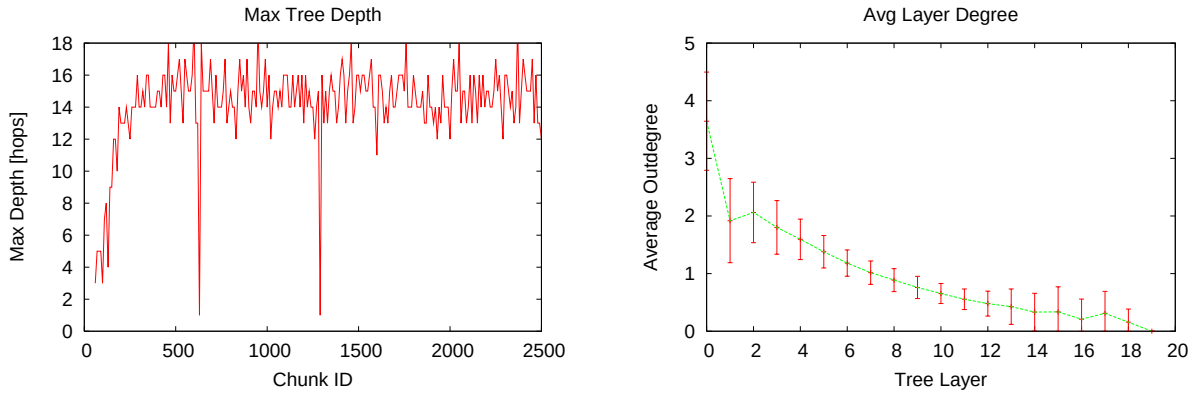


Figure 4.9 : Chunk distribution tree properties for HO-LB scenario with 500 peers and $u_s = 4\ SR$.

run different experiments with increasing number of peers from $n = 70$ to $n = 500$, and a source with upload capacity $u_S = 2\ SR$ and $u_S = 4\ SR$.

As in the previous scenarios, we observe in Figure 4.9 that the length of the different distribution trees is almost constant for all chunks. The source, corresponding to layer 0, distributes every chunk to 4 different peers if its nominal upload bandwidth is set to $u_S = 4\ SR$. Starting from level one, chunks are re-distributed by peers with upload capacity close to the stream rate ($1.1\ SR$). Peers belonging to level 1, 2 and 3 show an average degree of about 2 while the ones belonging to level 4, 5 and 6 have an average degree between 1 and 2. The remaining peers present an average degree of 1 or less. In such bandwidth conditions, *the first levels of distribution trees behave on average like a binary tree while, going down to leaves, the number of peers having just one child increases*.

However, peers have an upload capacity close to the stream rate and intuitively they should upload on average one copy of each chunk. To explain this behavior, we analyze the position of nodes in different distribution trees. We notice that a peer is an interior node for 55% of chunk distribution trees and a leaf in the remaining 45%. A peer rarely stays in the same layer for two consecutive chunk-trees (just in 10% of cases), and is such situations the peer is usually at the bottom of the considered trees. In most cases, a peer uses its available bandwidth to distribute two times the same chunk and it is a leaf in the distribution tree of the following chunk. A peer can also distribute one copy of two consecutive chunks or, less frequently, it can be a leaf for two consecutive chunks. So PULSE's distribution trees show properties similar to a *multiple tree of degree two*, particularly in their upper layers, and also respect their disjointness property. Moreover, if we look at peers' load distribution, we notice it is close to the stream rate for all nodes involved in the experience. This is another important characteristic of multiple tree structures which assure the load is *fairly distributed among nodes*.

In order to explain why distributions trees behave like binaries trees it is necessary to consider the chunk distribution mechanism. When a chunk A is injected in the system by the source, it is rare and it is owned by peers with a small buffer delay. Chunk A is required by lot of nodes, so owner peers use their upload bandwidth for its diffusion. Before chunk A has reached all the peers, a new chunk B is injected in the system by the source and becomes the rarest one. However, peers uploading chunk A cannot upload chunk B because they are already using their

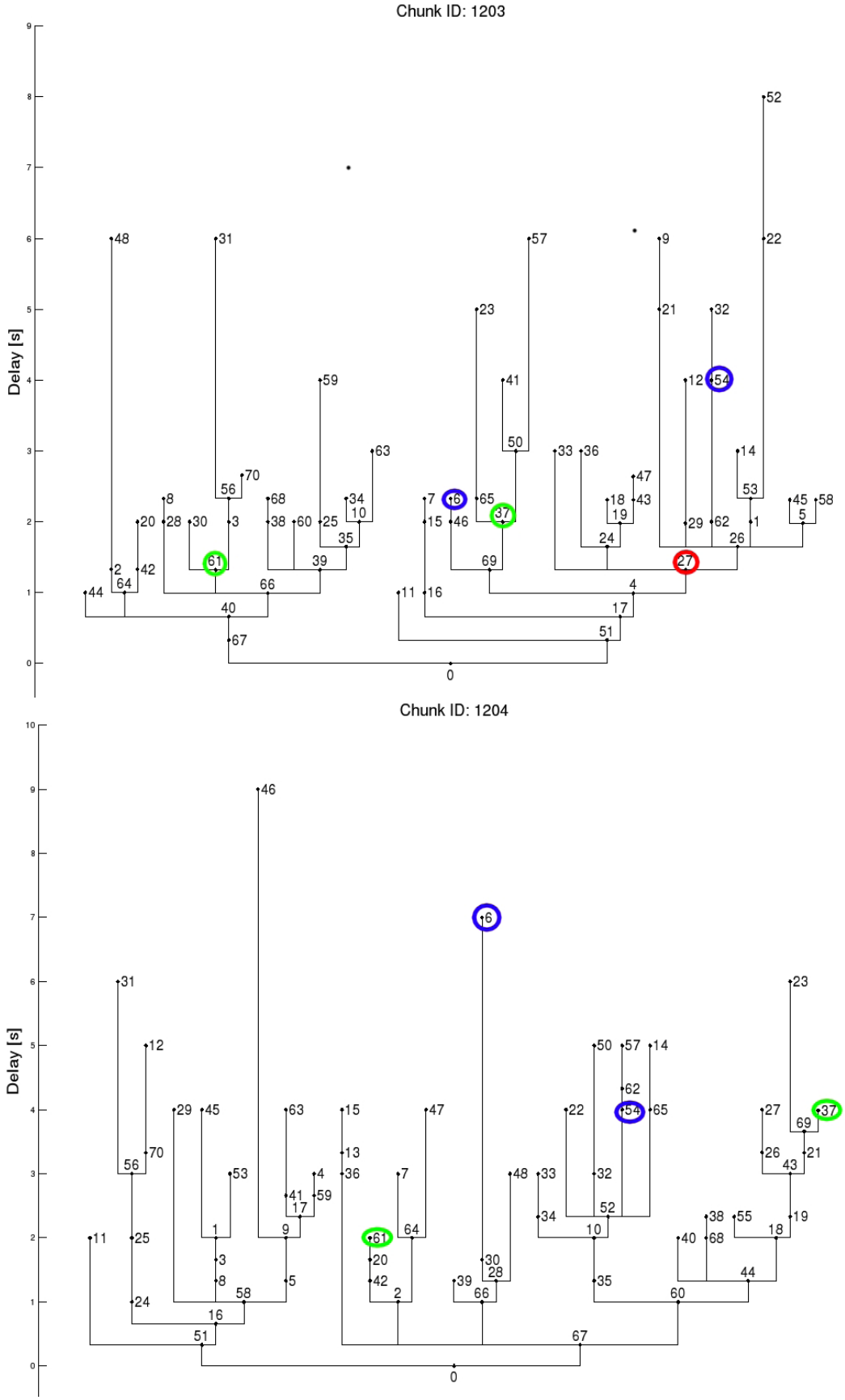


Figure 4.10 : Chunk distribution trees for HO-LB scenario with 70 peers and $u_s = 2\ SR$.

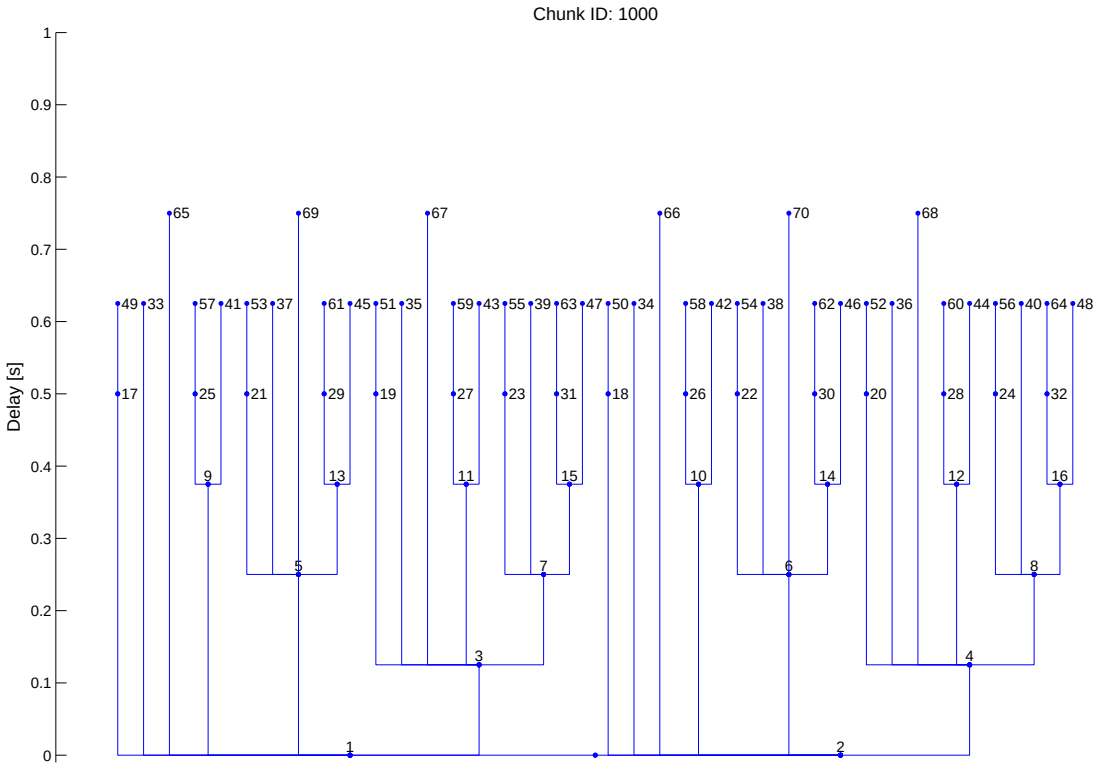


Figure 4.11 : Optimal distribution tree for 70 peers and $u_s = 2\ SR$.

bandwidth capacity. Thus other peers with small delay and free upload bandwidth should do that. After a few moments chunk A and B are both partially diffused in two binary sub-trees (sub-tree A and sub-tree B). Now peers of sub-tree A send the chunk A to peers of sub-tree B and vice versa. Peers of sub-tree B (A) receiving chunk A (B) can send it to upper layers of their sub-tree. This explains why distribution trees present a first part similar to a binary tree and a second part more chaotic. The same results have also been obtained in [69] where the two distribution phases are called *diffusion* and *swarming*.

Figure 4.10 shows the distribution trees of two consecutive chunks; peers receiving the chunk with the same delay are plotted on the same row and thus peers of the same distribution layer can be on different rows. We observe the behaviors explained before but also some particular phenomena. As said before, PULSE's distribution trees are not built by following precise criteria but are the results of local exchanges driven by local knowledge. As a consequence, we can just derive some general and averaged trends instead of precise statistics, because particular behaviors can be found on every tree. For instance in Figure 4.10, peer 61 and peer 37, are central nodes of degree 2 for chunk 1203 and leaves for chunk 1204. This is the typical behavior of nodes belonging to multiple-trees of degree two. However other different behavior can be observed. Peer 6 is twice a leaf probably because its buffer delay is big, while peer 54 just distributes once both chunks. Node 27 distributes chunk 1203 tree times; this can be explained by the fact that the third copy is made with 1.5 seconds of delay and thus node 27, has newly available bandwidth to distribute again this chunk.

By decreasing the source bandwidth capacity, we could expect important degradation on the system performance. But this is not the case: if we half the source capacity, the delay only increases of less than one second over seven (from $u_S = 4\ SR$ to $u_S = 2\ SR$).

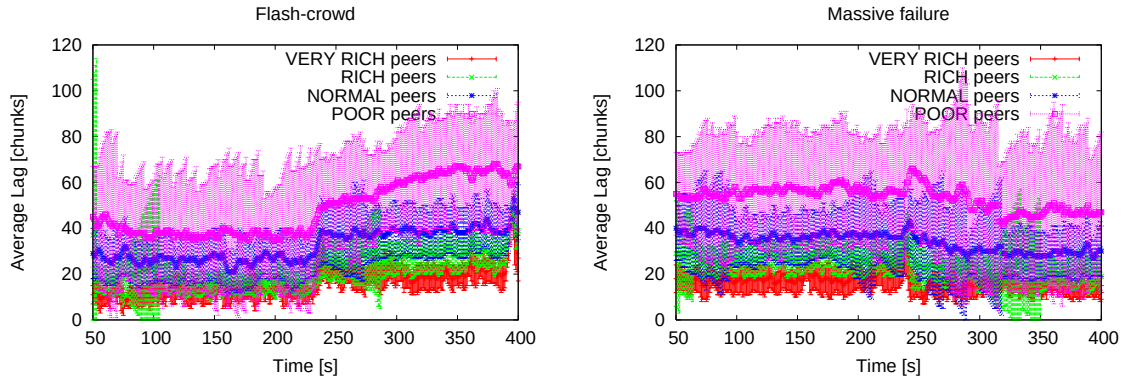


Figure 4.12 : PULSE under churn

Figure 4.11 reports the optimal distribution tree for $n = 70$ peers and one source of upload capacity $u_s = 2\ SR$. It is possible to observe the trees obtained by the chunk exchanges of PULSE, reported in Figure 4.10, are very different from this optimal one. In the following chapter, we are going to consider distributed chunk exchange algorithms that can achieve optimal or near-optimal diffusion rate/delay.

4.3.4 PULSE under Churn

We now analyze how PULSE behaves in presence of massive churn. In particular, we consider 2 kinds of churn applied over the HH-LB bandwidth scenario :

- *Flash-crowd*. Only 20% of the nodes joins the system at the beginning of the experiment (the proportion among the different classes is respected). After $t = 230\ s$ the remaining 75% are injected into the system.
- *Massive failure* All nodes join the system together at the beginning of the experiment. After $t = 270\ s$ we suddenly kill 50% of the peers (again, proportionally chosen).

We can observe in Figure 4.12 how the system handles gracefully both circumstances, with a noticeable increase/decrease of T_B over the seconds immediately following the arrival/departure that quickly leads to new stable average lag values. Visually, the effect of these massive churn events is completely absorbed after about 150 seconds from their occurrence. Also, as in all previous traces, no nodes ever suffered any data loss which could have led to playback disruption.

4.3.5 Brief comparison to other systems

In this section we present a comparison between the diffusion delay achieved by PULSE and the diffusion delay theoretically achieved by structured systems. For a fair comparison, we use the PULSIM simulator developed by Fabio Pianese [89]; experimental results may be affected by practical factors that cannot be taken into account by the theoretical analysis of structured

systems. We run simulations with $n = 500$ nodes having upload bandwidth equals to the stream rate, a source with upload capacity $u_S = 4\ SR$ and all the other parameters are set as in the Grid5000 experiences (Table 4.2). This scenario misses lot of good design properties of PULSE but is needed to compare it to structured systems, which are mainly designed for homogeneous upload capacity.

Remember that in PULSE chunks are stored in a buffer whose lag may vary over time. As delay metric we consider the *biggest* lag observed among all nodes during the whole simulation time. As already observed in previous experiments the delay achieved by peers is almost constant over time with a peak of 2.5 seconds. For structured systems the reception delay can easily be

Protocol	Maximal delay d=2 [s]	Maximal delay d=4 [s]	Buffer overhead [s]
Optimal	1.18		-
Single tree	2.24	2.24	0
Multiple tree (SplitStream)	2.24	2.24	1/SR
Cluster tree	2.24 ($m_c = 2$) 2.61 ($m_c = 5$)	2.61($m_c = 4$) 2.86 ($m_c = 9$)	(m_c-1)/SR
ZigZag	4.48 ($m_c = 3$) 71 ($m_c = 9$)	2.24 ($m_c = 3$) 35.86 ($m_c = 9$)	0
PrefixStream	2.49($m_c = 2$) 3.24($m_c = 8$)	2.49($m_c = 2$) 4.61 ($m_c = 16$)	(m_c-1)/SR
PULSE	2.5		4.0 (2 W_S)

Table 4.4 : Comparison between PULSE and existing protocols. m_c denotes the cluster size.

computed because of the mathematical properties of data delivery paths. So it is not necessary to implement them in order to derive their performance. The values contained in Table 4.4 are computed by using the analysis presented in [42] by setting the parameters according to our simulative environment.

Table 4.4 shows that PULSE, with a delay a bit more that *twice the optimal delay*, performs well with respect to structured protocols.

4.4 PlanetLab Deployment

PlanetLab [92] is a platform composed by machines spread worldwide. It has been developed by a consortium of universities and research institutes in order to provide a testbed for network applications in a real world environment. Every associated member provides a certain number of machines and a user can run experiments on all the active ones. The interconnections are not dedicated links, but the traffic among PlanetLab nodes is routed through normal Internet links. The machines are not under the control of a particular user but lot of people can perform tests using the same machines at the same time.

A recent study [11] highlights that PlanetLab nodes are not representative of typical connectivity of Internet hosts, as they are placed along well-provisioned access points, and their network diversity is lower than the average diversity of commercial Internet. Such bandwidth is however not completely exploitable because of the aforemetioned load problems on machines, so that the actual capacity of nodes is hard to predict or monitor.

The high CPU load and unpredictable bandwidth problems make it especially difficult to test a time-sensitive streaming application that requires low response times. For this reason, we perform a pre-selection of about 200 hosts with semi-acceptable CPU load conditions, while we have to lower the chunk rate to $\lambda = 4$ chunks per second and the stream bitrate to $SR = 128$

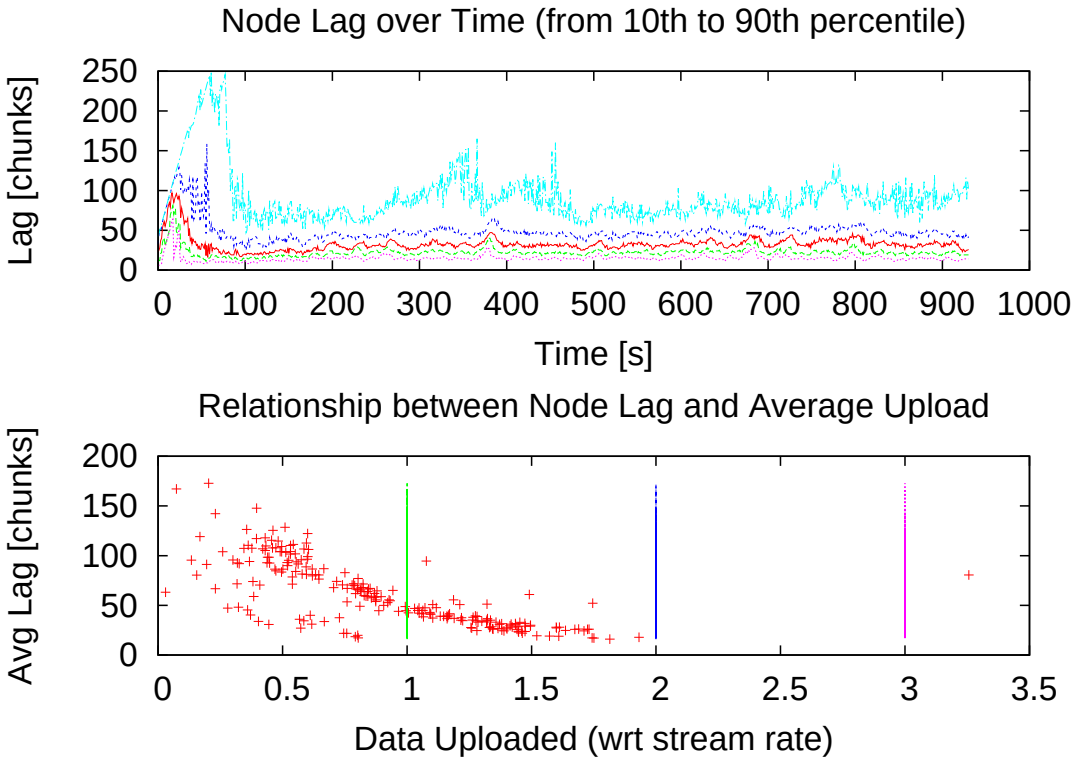


Figure 4.13 : PULSE over PlanetLab

Kbps. We do not limit the node bandwidth but chose to leave it naturally limited by the resources available at each host, since the high CPU load in most PlanetLab nodes would also slow down the execution of the software in unpredictable ways.

The results, reported in Figure 4.13 show that the use of an incentive-based selection allows PULSE to behave reasonably well even on this difficult environment, proving a high level of resilience and adaptiveness. It can be noticed that 90% of peers manage to regularly obtain a T_B lower than about 100 chunks (25 seconds), and that 50% present a node lag lower than 30 chunks (less than 10 seconds). The T_B distribution is a consequence of the upload bandwidth distribution, as about 60% of peers offer less than the full stream rate while the other 40% upload at a rate lower than twice the stream rate. By correlating the total bandwidth contribution with the average lag of the nodes, we observe a clear inverse relationship between the two variables: the more a peer uploads, the lower is its lag.

About Latency awareness

We now consider the impact of a weighted latency bias on the system in terms of locality awareness. These experiments are performed on Planetlab using a population of 100 peers without any artificial upload limitation. We observe the behavior of PULSE as the W_{RTT} latency weight parameter is set to 0 and 1. Remember that the PULSE client estimates latencies towards its neighbors by means of RED messages. In our experience we measure the pairwise node latencies by using exponentially-spaced *ping* probes ($\lambda_{ping} = 10s$).

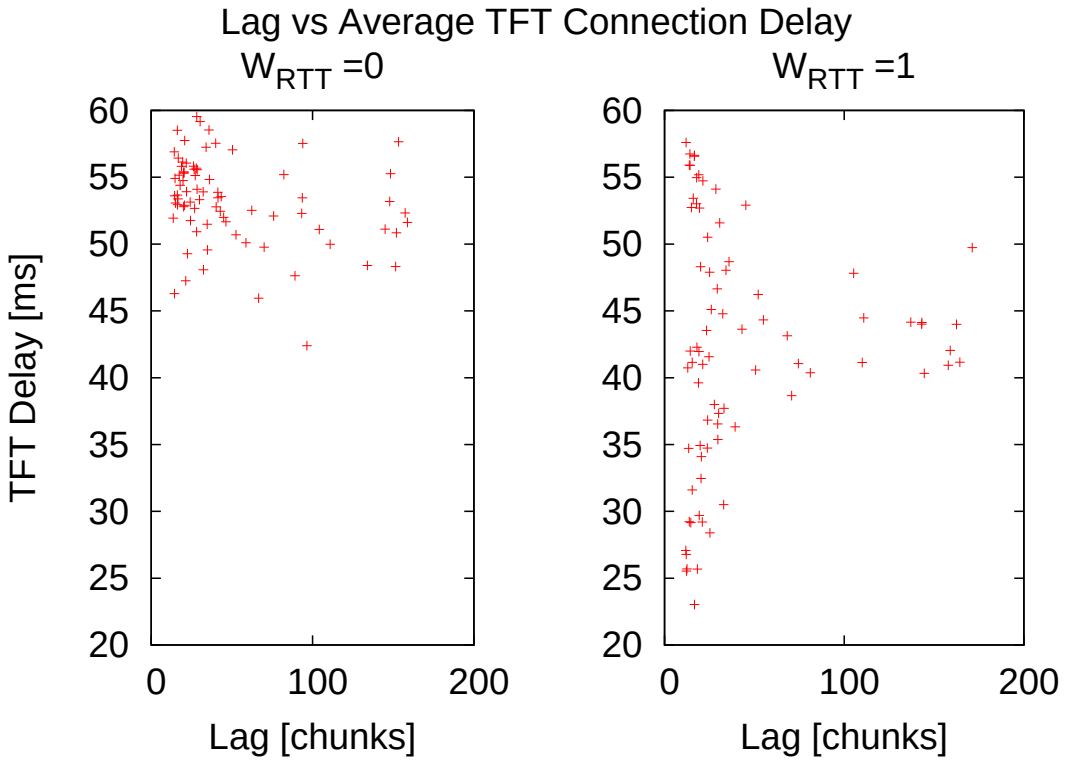


Figure 4.14 : Effect of latency bias on cumulative connection latency

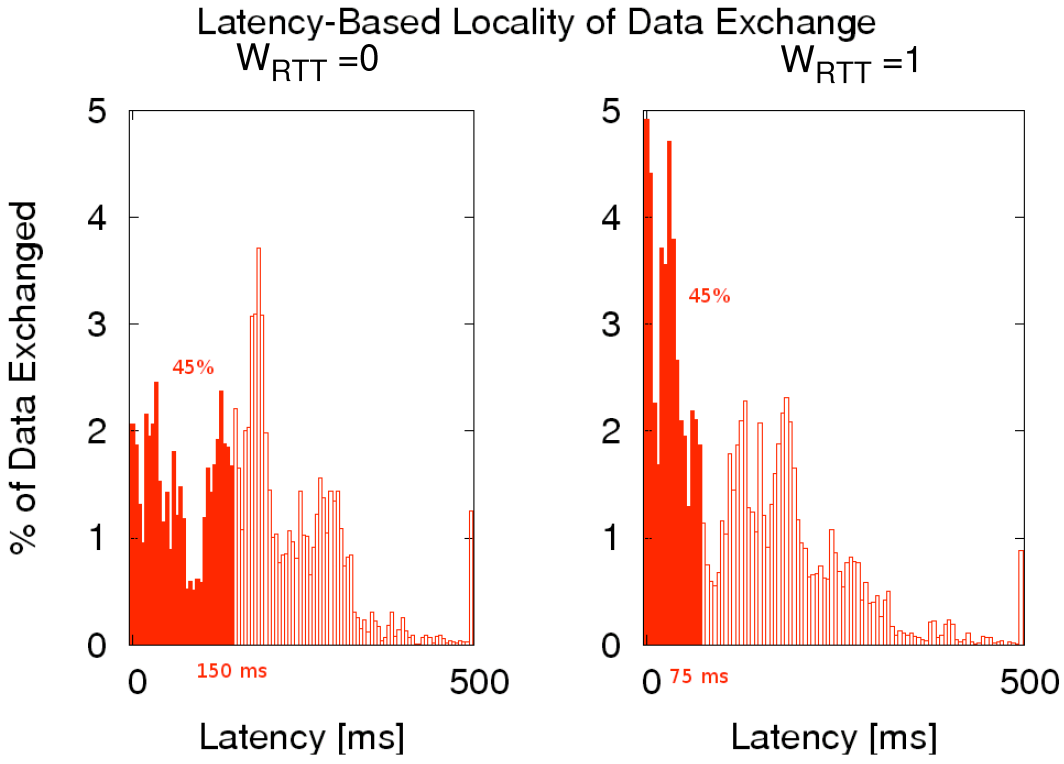


Figure 4.15 : Effect of latency bias on overall data exchange locality

Lag Percentile	10%	30%	50%	70%	90%
W_{RTT}=0	12.31	18.10	26.18	37.70	61.08
W_{RTT}=1	10.53	14.47	18.89	27.22	49.39

Table 4.5 : Effect of latency bias on average node lag (in chunks)

Figure 4.14 shows the *cumulative latency* of the incentive-driven connections as a function of the average lag of each peer. Cumulative latency is computed for each single node by adding together the latencies of the four connections that it established using the biased TFT incentive, and averaging this value over time. It is possible to notice how the introduction of the latency bias can sharply reduce the average TFT delay, especially for those peers whose lag is low. We can see that, when $W_{RTT} = 0$ (i.e. with no latency bias), all peers show an cumulative latency uniformly distributed between 45 ms and 60 ms, regardless of their average lag. With the addition of a latency bias $W_{RTT} = 1$, the minimum cumulative latency goes down to 22 ms, while just few nodes maintain a cumulative latency of about 60 ms.

In Figure 4.15 we correlate the percentage of data being uploaded by each peer with the latencies of the connections that it is using, again averaged over the time. The histograms clearly show that locality of data exchange definitely increases if we add a latency bias: when $W_{RTT} = 0$, the data is sent to other peers in an almost uniform way (we remember that the latency distribution of the peers is not uniform). On the contrary, when $W_{RTT} = 1$, the amount that has to travel on shorter distances is much higher: the stream data are prevalently exchanged between peers with pairwise latencies lower than 125 ms. Finally, Table 4.5 shows the effects of latency awareness on the global performance of data reception in the system, in terms of percentile node lag. As we expected, with the latency bias peers achieve a slightly lower reception delay, thanks to the fact that chunks are sent more often to peers which are *closer* locality-wise. The extent of this reduction is quite small, however, as the skew in the node latency distribution is quite limited in PlanetLab. We expect that, by introducing the latency bias in a scenario with larger difference between pairwise node latencies, the reception delay reduction could be more significant.

4.5 Conclusion

In this chapter we have presented and analyzed PULSE, a peer-to-peer mesh-based live streaming system which relies on a *tit-for-tat* incentive mechanism as its main peer selection policy.

We have shown the *mesh-incentive* approach is able to meet the live streaming requirements, and we have highlighted its flexibility and adaptability to network conditions and node resources. In particular, it turned out nodes contributing with more resources to the system are advantaged by achieving lower reception delays. The placement of these more resourceful nodes at the top of distribution trees also improves the global system performance by reducing the reception delays of all nodes thanks to short and first level width distribution trees. Moreover, a simple latency bias can strongly localize data exchange reducing the burden for the core of the network .

A deeper analysis of chunk distribution trees has shown data paths resulting from local exchanges have some emerging characteristics that are not so far from the design properties of structured systems. This allows PULSE to achieve performance that is comparable to tree-based applications in term of playout delay. However, PULSE diffusion delay is still more that

twice the optimal one: this is a consequence of the randomness and locality of the PULSE protocol, while optimal diffusion requires more deterministic algorithms.

Chapter 5

Epidemic live streaming

In the past few years several commercial live streaming systems have been proposed. Measurement studies highlight that the most popular applications rely on mesh-based approach for the stream distribution, and confirm the effectiveness of this approach for the deployment of large-scale live streaming systems over the Internet. Despite their popularity, the fundamental mechanisms of data dissemination in such systems cannot be completely analyzed because allocation algorithms are not disclosed.

In the previous chapter we have analyzed the performance of the allocation algorithms of PULSE, a mesh-based live streaming systems which rely on *tit-for-tat* peer selection strategy and *local rarest first* chunk selection policy. This application has been designed and developed by our team, it is open source and so we can completely control and understand its behavior. Results highlight the effectiveness of the resource aware-mesh approach for deployment of live streaming systems. However, they also show the diffusion delay obtained by nodes is more than twice the optimal theoretical one as a consequence of non-optimal chunk distribution trees. Moreover, it is difficult to derive a model of PULSE because of the complexness of the whole system.

In this chapter we tackle the problem from a more theoretical perspective. In order to fully understand the stream dissemination process in mesh-based systems, we focus on chunk/peer selection policies only, and we disregard from other issues. In particular, we consider epidemic-style distribution algorithms where every chunk forward is the result of a chunk/peer selection performed locally at one node.

The goal of the approach is to address questions like: is it possible to achieve optimal or near-optimal rate/delay performance in mesh-based live streaming systems? Which are the main performance trade-offs of such algorithms?

We first propose an overview of optimality results concerning diffusion rate and delay in Section 5.1. In Section 5.2, we design some practically interesting resource allocation schemes for homogeneous systems; we derive recursive formulas to describe the diffusion process of some of them and we provide some simulative results for a better understanding of the diffusion mechanisms.

In Section 5.3, we modify the proposed selection schemes in order take into account the resources shared by nodes in heterogeneous systems. We adapt our recursive formulas to model such aware diffusion schemes, and we deeply analyze the awareness/agnostic trade-off in the

peer selection policy. Finally in Section 5.4, we highlight the important role critical parameters, like chunk size and neighborhood size, play in the algorithms performance.

Contents of this chapter are a joint work with Thomas Bonald, Nidhi Hegde, Laurent Massoulié, Fabien Mathieu and Andy Twigg and are partially presented in [141, 134].

5.1 Optimal diffusion schemes

In this section, we focus on diffusion schemes whose performance have been proven optimal in specific scenarios. In particular, we consider performance optimality in terms of diffusion rate and diffusion delay. We only consider effective data transfers, and we do not take into account other overheads like control messages, protocol overheads and so on.

In live streaming, as for all other fixed-rate applications, the maximal goodput d achieved by a node cannot be larger the content generation rate i.e. the stream rate SR (but the throughput can)[1]. If the number of leechers is higher than the one sustainable by the system, the throughput is necessary lower than the stream rate for some peers. Please refer to Chapter 2 for more details.

We say a diffusion scheme is *rate optimal* if it perfectly exploits the available bandwidth. In other words:

- the overhead is small, so goodput and throughput are almost the same;
- if there are enough resources in the system, then $d = SR$ for all nodes;
- if not, the rate optimality coincides with a maximization of the throughput, i.e. the sum of peers' throughput is equal to the total available bandwidth.

In particular, in a scenario where the bandwidth shared by every node is on average equal to the stream rate, a scheme achieving optimal diffusion rate is able to provide every chunk to every peer.

Concerning delay optimality, it has ben shown that in an homogeneous system, where all peers provide an amount of bandwidth equal to the stream rate, the minimal diffusion delay is $log_2(i) + 1$ time units[2], where i is the number of peers receiving a given chunk. Therefore, we say a scheme is *delay optimal* if it the last peer receiving a given chunk gets it in $log_2(i) + 1$ time units after it has been generated by the source. The minimal diffusion delay in heterogeneous systems is more complex, but some bounds have been proposed [65, 75].

There is a natural trade-off between diffusion rate and delay. The diffusion rate is typically maximized by a homogeneous dissemination of chunks among peers, irrespective of the age of these chunks. However, such age agnostic dissemination may lead to high diffusion delays. On the other hand, to minimize the diffusion delay, priority should be given to the transmission of the more recent chunks rather than to the homogeneous dissemination of chunks among peers. The price to pay is a sub-optimal diffusion rate because chunks are not retransmitted anymore by a given peer once it has received fresher ones.

[1] Remember that the throughput is the bandwidth used by peers, while the goodput is the actual data received.

[2] In such an homogeneous context a time unit is the time needed by a peer to upload a chunk, and all peers upload exactly one chunk per time unit.

Zhang et a. focus on throughput optimization. In [128] they propose an optimization framework to model the chunk scheduling problem and they derive a min-cost flow formulation to solve it in polynomial time. This solution is centralized and therefore not applicable in peer-to-peer systems, but they derive a sub-optimal distributed heuristic based on a local optimal chunk scheduling performed at every node.

In [129], they prove pull-based protocols can achieve near optimal capacity utilization and throughput. This optimality strongly depends on parameter settings, and an important trade-off between control overhead and diffusion delay emerges. To improve performance, they propose an hybrid push-pull protocol but they do not provide optimality results for it. In a nutshell, the protocol pushes packets along the near-optimal diffusion trees formed by the pull technique.

Massoulié et. al. [73], prove the rate optimality of the so-called *most deprived peer/random useful chunk* algorithm, and Sanghavi, Hajek and Massoulié [100], prove the delay optimality of the *random peer/latest blind chunk* algorithm (these algorithms are detailed further in this chapter). It turns out, however, that the delay performance of the former is poor due to the random chunk selection, while the rate performance of the latter is poor due to the random peer selection.

Our work [141] is the first to prove that a scheme, the *random peer/latest useful chunk* algorithm, can achieve optimal diffusion rate within a delay of $log_2(n) + O(1)$ where the $O(1)$ additive term, is a random variable bounded in probability uniformly in the number of peers n. However, we are going to highlight in Section 5.2.3 that, when the system is close to critical regime, this additive constant may be significant and other schemes can achieve optimal rate dissemination in shorter times. This has also been shown by Zhou et al. in [132] where they derive recursive formulas to describe the diffusion functions of *latest useful* and *earliest useful* chunk selection policies, and of a *mixed latest/earliest strategy*. They show latest useful is not optimal for any given delay and that the mixed strategy can achieve a better diffusion rate within the same delay.

In [141] we also show, the *random peer/latest blind chunk* algorithm can achieve optimal diffusion rate too, if coupled with source coding. Such a diffusion scheme is known to achieve a diffusion rate of only $1 - e^{-1}$ in the critical regime where the source speed is equal to the upload speed [100]. It is thus necessary to add some redundancy to the original signal to allow the peers to recover from chunk losses. We show that the additional delay due to the coding/decoding scheme can be controlled (that is, made be equal to O(1)) by bounding the correlation of successive missing chunks.

More recently, Abeni, Kiraly and Lo Cigno [3] prove there exists a diffusion scheme that can distribute a chunk to n peers in exactly $log_2(n) + 1$ time units. This scheme, first selects the chunk by means of a *deadline-based* chunk selection policy, and then the peer by means of an *earliest-latest* peer selection policy.

An interesting survey on optimality results an open questions about optimal diffusion schemes can be found in [71].

5.2 Algorithms for homogeneous bandwidth systems

We start our analysis by considering schemes that run on homogeneous environments, where all peers have the same upload capacity. For these scenarios, where there is no need to take

into account the respective resources of the nodes, we consider some simple, yet practically interesting, diffusion schemes and we analyze the rate/delay trade-offs they achieve by means of simulations and recursive formulas.

5.2.1 Model and algorithms

We consider a P2P system of n peers and a single source S. The source creates a sequence of chunks, numbered $1, 2, 3, \ldots$, at rate λ (expressed in chunks per time unit), and sends each chunk to one of the n peers, chosen uniformly at random. L is the set of peers so that $|L| = n$. For any $l \in L$, we denote by $u(l)$ the upload speed of peer l. This is the maximum number of chunks that l can *send* per time unit. For simplicity, we assume that there is no constraint on the number of chunks that each peer can *receive* per time unit. In this section we suppose a common upload speed $u(l) = 1$ for all peers.

We say that the system is in *underload* regime if $\lambda < 1$, in *critical* regime if $\lambda = 1$ and in *overload* regime if $\lambda > 1$. Clearly, some peers receive only a fraction of the chunks sent by the source in the overload regime. Nevertheless, peers may successfully decode the original audio or video streaming signal if some redundancy has been added to this signal and is included in the chunks sent by the source. Thus all three regimes are of practical interest.

We suppose the sender initiates the transmission of a chunk between two peers , so that we formally focus on *push-based* diffusion schemes. However, the difference between push and pull schemes is not that relevant in our framework, as we only care about the resulting data exchanges. We assume each peer has only a partial knowledge of the overall system. This is represented as a directed graph $G = (L, E)$ where $(l, v) \in E$ if and only if l knows v, for all $l, v \in L$ (we say that l is a neighbor of v). We denote the set of neighbors of peer l as $N(l)$ and we suppose a peer can only send chunks to one of its neighbors.

For any $l \in L$, let $B(l)$ be the collection of chunks that peer l has received. We denote by $\mathcal{B}$ the set of possible collections of chunks owned by a peer. A push-based scheme is formally described as a (possibly random) mapping from $L \times \mathcal{B}^n$ to $L \times \mathcal{B}$ that gives for any sender peer l, as a function of the collections of chunks $B(v)$ of its neighbors v, the destination peer and the chunk $b \in B(l)$ to be sent.

Diffusion schemes may be categorized into two classes depending on whether the destination/sender peer or the chunk is selected first. In this section, we shall restrict the analysis to the following push based peer and chunk selection schemes:

Random peer: The destination peer is chosen uniformly at random among the neighbors of l;

Random useful peer: The destination peer is chosen uniformly at random among those neighbors v of l such that $B(l) \setminus B(v) \neq \emptyset$. When the chunk b is selected first, the choice of the destination peer is restricted to those neighbors v of l such that $b \notin B(v)$;

Most deprived peer: The destination peer is chosen uniformly at random among those neighbors v of l for which $|B(l) \setminus B(v)|$ is maximum. When the chunk b is selected first, the choice of the destination peer is restricted to those neighbors v of l such that $b \notin B(v)$;

Latest blind chunk: The sender peer l chooses the most *recent* chunk (that is, the chunk of highest index) in its collection $B(l)$;

Latest useful chunk: The sender peer l chooses the most recent chunk b in its collection $B(l)$ such that $b \notin B(v)$ for at least one of its neighbors v. When the destination peer v is selected first, b is the most recent chunk in the set $B(l) \setminus B(v)$.

Random useful chunk: The sender peer l chooses uniformly at random a chunk b in its collection $B(l)$ such that $b \notin B(v)$ for at least one of its neighbors v. When the destination peer v is selected first, b is chosen uniformly at random in the set $B(l) \setminus B(v)$.

A rich class of push-based schemes follows from the combination of these peer/chunk selection algorithms. Those considered in the section are summarized in Table 5.1. Figure 5.1 gives an example of peer/chunk selection under these schemes. Note that, for this particular example, the latest chunk of the sender peer has already been received by all its neighbors. The transmission capacity of the sender peer is then wasted in this state under the *lb/up* and *lb/rp* schemes, since a peer will receive two or more copies of the same chunk.

Notation	Scheme
rp/lb	random peer/latest blind chunk
rp/lu	random peer/latest useful chunk
dp/lu	most deprived peer/latest useful chunk
dp/ru	most deprived peer/random useful chunk
lb/rp (= *rp/lb*)	latest blind chunk/random peer
lb/up	latest blind chunk/useful peer
lu/up	latest useful chunk/useful peer
lu/dp	latest useful chunk/most deprived peer

Table 5.1 : Some push-based diffusion schemes.

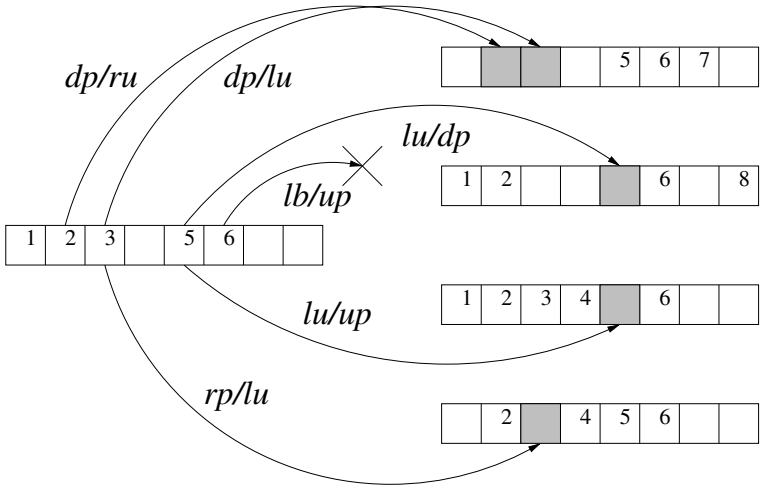


Figure 5.1 : Peer/chunk selection of a sender peer (left) under the considered push-based schemes.

In this section, we assume that time is slotted so that the transfer of any chunk by any peer takes exactly one time slot. The source sends $\lfloor \lambda \rfloor$ chunks per time slot, plus one additional chunk

with probability $\lambda - \lfloor\lambda\rfloor$, corresponding to an arrival rate λ. Note that for $\lambda < 1$, the source sends chunks according to a Bernoulli process.

In a first approach, we assume that at each slot, every peer has a perfect knowledge of the state of its target peer, including the intended transmissions of other peers to the same target peer. In particular, all conflicts are solved at the beginning of each slot, prior to the chunk transmission. The impact of imperfect knowledge resulting in transmissions of the same chunk to the same target peer will be analyzed for the example of the lb/ru scheme in Section 5.2.2.

5.2.2 Recursive formulas

In this section, we derive recursive formulas for the diffusion function of the *latest blind chunk/random peer* and the *latest blind chunk/random useful peer* schemes through mean-field approximations. Under the former, each peer simply sends the latest chunk it has to a randomly chosen peer; under the latter, it sends the latest chunk it has to a randomly chosen peer among those peers that have not yet received this chunk, if any.

We consider a reference scenario with complete graph. We assume that $\lambda \leq 1$; the overload regime $\lambda > 1$ is considered in the last paragraph. The number of peers n is assumed to be sufficiently large so that the system may be considered in the mean-field regime where peers are mutually independent. We further assume that, for any given peer l, the event that a chunk belongs to the collection $B(l)$ of chunks owned by l is independent of the event that any other chunk belongs to $B(l)$. The validity of the derived formulas will be assessed by comparison with simulations in Section 5.2.3.

Beside the results presented in this section, other two papers propose recursive formulas for the diffusion functions of allocation algorithms. In [132], the *latest useful*, the *earliest useful* and the mixed *latest/earliest* chunk selection policies are analyzed. In [21] the diffusion functions of *random useful peer/random useful chunk*, *deprived peer*[3]*/random useful chunk*, *latest blind chunk/random useful peer*, and *latest useful chunk/random useful peer* are analyzed in case of a limited number of neighbors and in presence of delayed buffer map updates and overlay churning.

Latest blind chunk / random peer

We first consider the *lb/rp* scheme. Recall that $r(t)$ corresponds to the average fraction of peers that receive any given chunk no later than t time slots after its creation. Without any loss of generality, we assume that some tagged chunk is created at time $t = 0$ and that the system is in steady state at that time. Since the source sends each new chunk to a randomly chosen peer, we have $r(1) = 1/n$. Now at any time $t \geq 1$, the tagged chunk is the latest of the collection owned by an arbitrary peer i with probability:

$$p(t) = r(t) \prod_{k=1}^{t-1} (1 - \lambda r(k)). \tag{5.1}$$

[3]Every sender peer l selects a neighbor v with a probability proportional to the number of useful chunks it has for peer v. This is different from the most deprived peer scheme where peer l selects the neighbor v for which it has the highest number of useful chunks.

This follows from the independence assumption, noting that for all $k = 1, 2, \ldots, t$, $r(k)$ is the probability that a chunk created at time $t - k$ is in the collection $B(i)$ of chunks owned by peer i at time t.

Due to the random peer selection strategy, the number of copies of the tagged chunk that are received by an arbitrary peer at time $t + 1$ is a binomial random variable with parameters $(n - 1, p(t)/(n-1))$. For large n, this can be approximated by a Poisson random variable with mean $p(t)$. Thus the probability that an arbitrary peer receives at least one copy of the tagged chunk at time $t + 1$ is approximately equal to $1 - e^{-p(t)}$. A fraction $1 - r(t)$ of the peers that receive the chunk at time $t + 1$ actually need it. We deduce the recursive formula:

$$r(t + 1) = r(t) + (1 - e^{-p(t)})(1 - r(t)), \quad t \geq 1, \tag{5.2}$$

where $p(t)$ is given by (5.1).

Latest blind chunk / random useful peer

We now consider the *lb/ru* scheme. The only difference with the *lb/rp* scheme is that all transfers are useful as long as some peers need the considered chunk. This gives the recursion:

$$r(t + 1) = r(t) + \min(p(t), 1 - r(t)), \quad t \geq 1, \tag{5.3}$$

where $p(t)$ is given by (5.1).

Delayed updates

As explained earlier, some control messages are needed to maintain a fresh view of the collection of chunks owned by each peer. Delaying some control messages reduce the overhead but may impact the performance of the system. We model such delayed updates by assuming that peers know the state of system in the previous slot, but are not aware of the ongoing transfers of the current slot. Therefore, collisions can occur even under the *lb/ru* scheme when several peers send the same chunk to the same target peer.

Consider the diffusion of the chunk created at time $t = 0$. A fraction $1 - r(t)$ of the n peers has not yet received this chunk at time t. Thus the number of copies of this chunk that are received by one of these $n(1 - r(t))$ peers at time $t + 1$ is a binomial random variable with parameters $(n - 1, p(t)/n(1 - r(t)))$, where $p(t)$ is given by (5.1). For large n, this can be approximated by a Poisson random variable with mean $p(t)/(1 - r(t))$. Thus the probability that a peer that has not yet received the considered chunk at time t receives at least one copy of this chunk at time $t + 1$ is approximately equal to $1 - e^{-p(t)/(1-r(t))}$. We deduce the recursive formula:

$$r(t + 1) = r(t) + (1 - r(t))(1 - e^{\frac{-p(t)}{1-r(t)}}), \quad t \geq 1. \tag{5.4}$$

Overload regime

In the overload regime, $\lfloor \lambda \rfloor$ new chunks are created by the source at each slot, plus one additional chunk with probability $\lambda - \lfloor \lambda \rfloor$. The diffusion processes of these $\lfloor \lambda \rfloor$ or $\lfloor \lambda \rfloor + 1$ chunks will interfere in the diffusion process. We number these chunks as $1, 2, \ldots, \lfloor \lambda \rfloor$ (or $\lfloor \lambda \rfloor + 1$),

where chunk 1 corresponds to the last created chunk. Thus chunk 1 has priority over chunk 2, chunk 2 over chunk 3, and so on.

Now let r_i be the diffusion function associated with a chunk of index i. Again, we assume that some tagged chunk of index i is created at time $t = 0$ and that the system is in steady state at that time. At any time $t \geq 1$, this chunk is the latest of the collection owned by an arbitrary peer u if u has got it and hasn't got any fresher chunk. This happens with probability:

$$p_i(t) = r_i(t) \prod_{j=1}^{i-1} (1 - r_j(t)) \times \prod_{k=1}^{t-1} \left((1 - (\lambda - \lfloor \lambda \rfloor) r_{\lceil \lambda \rceil}(k)) \prod_{j=1}^{\lfloor \lambda \rfloor} (1 - r_j(k)) \right). \quad (5.5)$$

There are now $\lceil \lambda \rceil$ recursive formulas, one per diffusion function r_i. These can be deduced from (5.2), (5.3), (5.4) by replacing the functions r and p by r_i and p_i, respectively, for each considered diffusion scheme.

The global diffusion function follows by averaging:

$$r(t) = \frac{1}{\lambda} \left((\lambda - \lfloor \lambda \rfloor) r_{\lceil \lambda \rceil}(t) + \sum_{i=1}^{\lfloor \lambda \rfloor} r_i(t) \right). \quad (5.6)$$

5.2.3 Simulation results

In this section, we evaluate the rate/delay performance trade-offs achieved by the push-based diffusion schemes of Table 5.1 by means of simulations.

Unless otherwise specified, results are derived for $n = 600$ homogeneous peers with a complete graph, which corresponds to an optimal diffusion delay of $\log_2(n) + 1 \approx 10$ slots. Chunks that arrive more than 50 slots after their creation are not taken into account, which is representative of a real live streaming system with limited playback delay. In particular, the diffusion rate is approximated by the value of the diffusion function $r(t)$ at time $t = 50$, and we consider as the delay the time needed to reach 95% of $r(t)$ (maximal delay with $\epsilon = 5\%$ as explained in section 3.1).

Reference scenario

We first consider a reference scenario that consists of a complete graph in the critical regime $\lambda = 1$ Figure 5.2 reports the corresponding diffusion functions. Recall that the time unit is the slot duration, which corresponds to the transmission time of a chunk by any peer.

In such a scenario, the simulations show that four of the considered schemed, namely *dp/lu*, *lu/dp*, *lu/up* and *lb/up*, clearly outperform the three others. The four of them achieve an optimal diffusion rate, and all but *dp/lu* show a diffusion delay very close to 10 slots. The *dp/lu* tends to be slower than the other three as a consequence of the priority given to the peer selection over the chunk selection. The performance of *dp/ru* and *rp/lb* schemes is good regarding either rate or delay but not both, as announced in Section 5.1. In fact, the delay performance of the former is poor due to the random chunk selection, while the rate performance of the latter is poor due to the random peer selection.

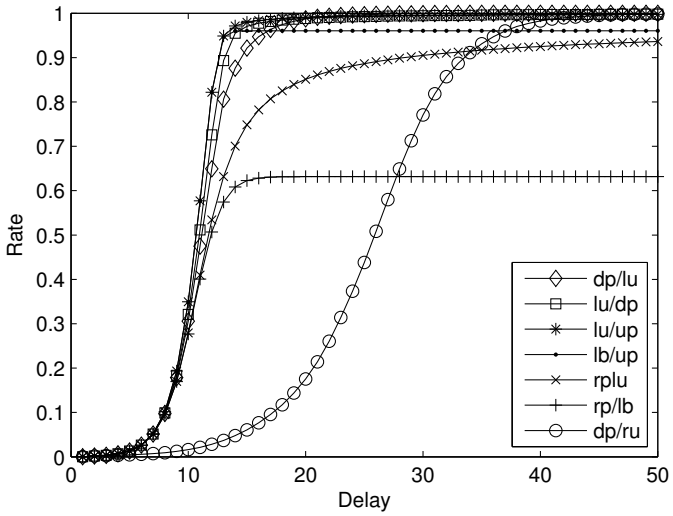


Figure 5.2 : Diffusion in the reference scenario.

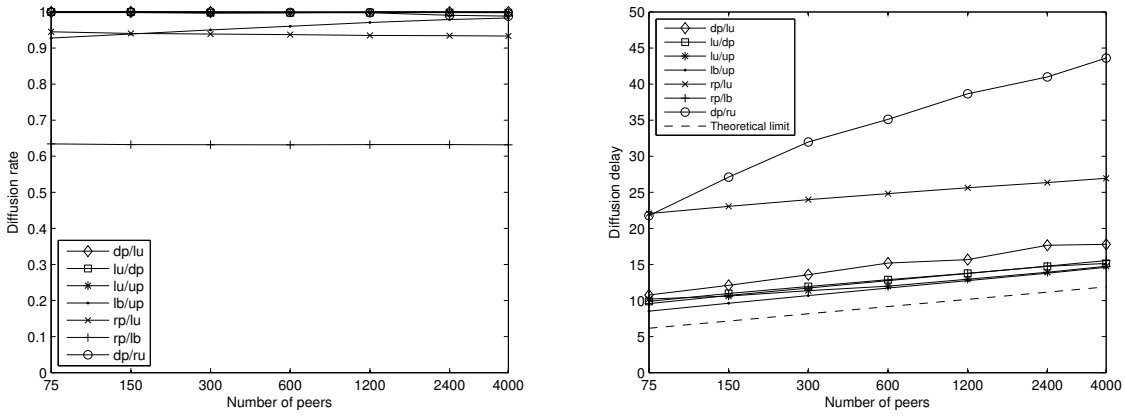
Finally, the *rp/lu* scheme tends to take a non negligible delay to achieve an acceptable rate, which may be surprising in view of the delay optimality of this scheme stated in Section 5.1. This is because the optimality result is not valid in the considered critical regime. Moreover, we shall see later that the additional constant delay predicted is significant even in the underload and overload regimes, as soon as the source speed λ is close to 1.

To summarize, we observe that the latest chunk selection policy can achieve near optimal diffusion delays, and, if it is coupled with a useful chunk selection, it can also achieve optimal diffusion rate. To select the peer first may reduce diffusion rate because, when the selection is performed, the sender peer is not sure to have useful chunks for the target peer. Again, this can be circumvented by selecting a peer for which there are useful chunks. However, useful peer/chunk selection first should require higher overhead because a fresher view of neighbor buffers is required with respect to a blind selection. A possible way to lower overhead is to reduce the neighborhood size as analyzed in the following.

Impact of the number of peers

We now let the number of peers n vary from 75 to 4000. The results are shown in Figure 5.3. The diffusion rate is constant for all schemes but the lb/up for which it increases with n, suggesting the asymptotic rate optimality of this scheme. As expected, the diffusion rate of rp/lb is equal to $1 - e^{-1}$. The rp/lu scheme, where the last useful chunk is selected, achieves a rate close to 0.93. All the other schemes achieve an optimal diffusion rate for all values of n.

All schemes but the dp/ru have an optimal diffusion delay of $\log_2(n) + O(1)$, which shows the good scalability of these schemes. The additional constant is significant for the rp/lu scheme (around 25 slots), moderate for the dp/lu scheme (between 5 and 10 slots), slight for the other schemes (less than 5 slots). Finally, the dp/ru scheme has poor delay performance, which is a consequence of the random chunk selection and induces a decrease of its rate for large n (some chunks are received after the maximum delay of 50 slots).



(a) Rate (b) Delay

Figure 5.3 : Impact of the number of peers.

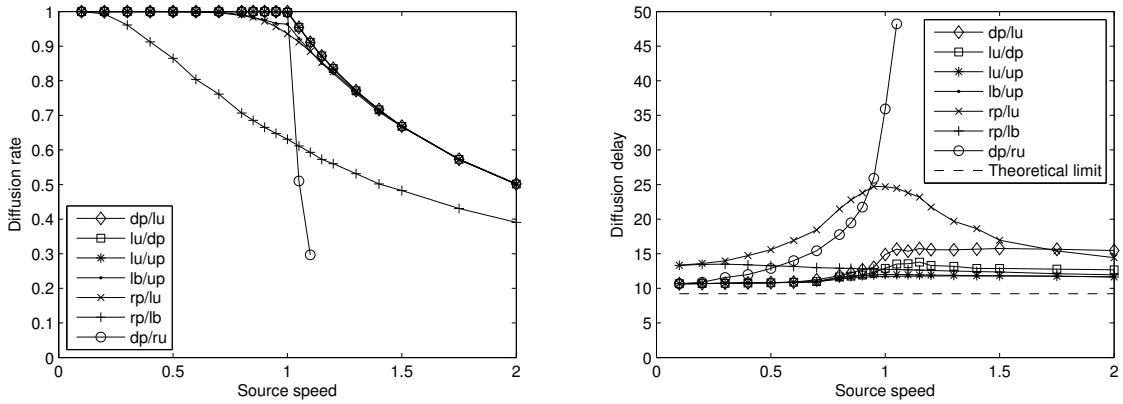
Impact of bandwidth provisioning

We now analyze the impact of the chunk generation rate λ; remember that λ is an indicator of the bandwidth provisioning of the system. Results are shown in Figure 5.4 when λ varies from 0 to 2, for $n = 600$ peers. Observe that the *rp/lu* scheme has poor delay performance not only in the critical regime $\lambda = 1$ but in all regimes close to critical, as announced. This means that the additional constant delay is far from negligible. The *rp/lb* scheme achieves a diffusion rate close to $1 - e^{-1/\lambda}$ in low delay, as expected [100].

The performance of the other schemes is nearly optimal for both rate and delay, except for the dp/ru and dp/lu schemes that behave poorly in overload regime. Note that the *dp/ru* scheme doesn't reach any steady state, which is a consequence of the random chunk selection coupled with the fact that each peer receives at most a fraction $1/\lambda$ of the chunks. Intuitively, in an overloaded regime, there are always old chunks to send, and *dp/ru* tries to send them; therefore the average relative age of chunks sent will grow linearly with the age of the stream, while a steady state would require that age to remain bounded.

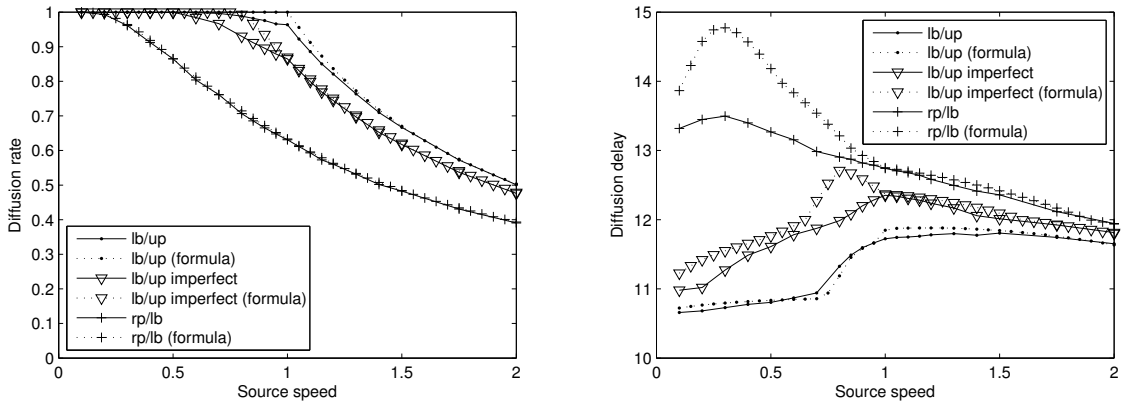
Validation of the recursive formulas

We now validate the mean-field approximation used to derive the recursive formulas of Section 5.2.2. Figure 5.5 compare the diffusion rate and diffusion delay obtained by analysis and by simulation, in a scenario where the source speed vary from 0 to 2. The formulas are quite accurate for both rate and delay. The most significant difference concerns the *rp/lb* scheme, where the formula overestimates the delay for $\lambda \approx 0.3$ by 1.5 slots (corresponding to an error of 10%). Regarding the *lb/up* scheme, the delay estimation is very good but the formula slightly overestimates the rate for λ close to 1 (the error is less than 4%). Finally, the formula of the *lb/up* scheme with imperfect knowledge slightly overestimates both delay and rate for $\lambda \approx 0.8$ (error less than 6% for both metrics). Interestingly, these anomalies occur at the maximum source speed λ for which the diffusion rate is very close to 1; this is due to the fact that the fraction of peers that need any given chunk at time t, approximated by $1 - r(t)$, becomes hard to estimate in this specific regime.



(a) Rate (b) Delay

Figure 5.4 : Impact of source speed.



(a) Rate (b) Delay

Figure 5.5 : Validation of the recursive formulas.

Restricted neighborhoods

A complete overlay graph presents a lot of practical issues: each node must be aware of all participants of the system (and it must be updated in case of arrivals or departures). Moreover, when a chunk exchange requires to know the current status of neighbors (this is for instance mandatory for *most deprived peer* schemes), the overhead burden may become prohibitive. On the other hand, for schemes like the *rp/lu*, where the blind peer selection reduce the need of fresh information to only one selected peer, or for completely blind schemes like the *rp/lb*, which do not require any information at all, the issue is lessened.

We propose to investigate three simple ways to bypass the overhead issue and make feasible the deployment of all the schemes presented in this chapter:

Static graph: To reduce the number of neighbors of every node. In particular, we consider an Erdös-Rényi graph with an average degree of 10, that ensures the graph is connected

with high probability for the considered set of $n = 600$ peers. The graph remains the same during the whole diffusion process.

Random graph: For each chunk transmission, the sender peer selects uniformly at random two peers among the $n - 1$ other peers; the diffusion scheme then applies to these two potential target peers. Note that the graph is now dynamic.

Adaptive graph: For each chunk transmission, the sender peer keeps track of the last target peer and select uniformly at random another peer among the $n - 2$ other peers; again, the diffusion scheme then applies to these two potential target peers. Note that this technique is somewhat reminiscent of the "optimistic unchoking" used by BitTorrent [30].

The impact of these techniques on scheme performance is shown in Figure 5.6 right side, in a scenario where the source speed λ varies from 0 to 2 and in a scenario where heterogeneity factor h varies from 0 to 1 (see Section 5.3 for details). The same instance of the Erdös-Rényi graph is used for all plots.

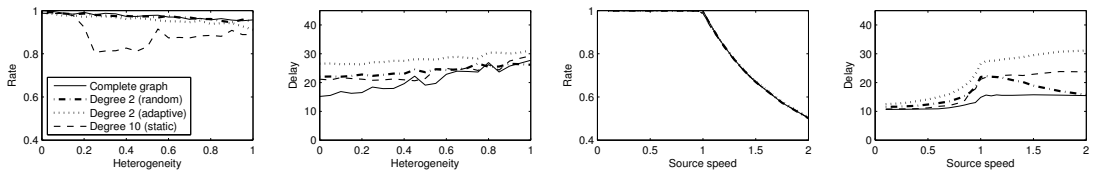
We observe that for most diffusion schemes, this static restriction of neighborhood strongly reduces the diffusion rate. The adaptive neighborhood, on the other hand, increases the diffusion delay of most schemes. Overall, it turns out that the basic random graph approach, where the sender peer selects two potential target peers at random, achieves the best trade-off. The performance degradation is slight in most cases compared to the complete graph. In particular, the top three schemes have very good performance, even in the worst case of heterogeneous networks in the critical regime. Moreover, the *rp/lu* behaves as in the complete graph case. This is not surprising because the peer is randomly selected first so that to reduce the set of potential recipients by another random selection cannot affect the performance.

From that perspective, the best compromise for real deployment in homogeneous systems is provided by selection policies like *rp/lu*, where a useful chunk selection is performed over only one peer. The drawback is that sometimes the selection of a peer may not be useful because the sender peer has not useful chunk for it. A possible solution is to select more than one peer in order to increase the probability to find a useful chunk. We better investigate the impact of the size of this probe set in section 5.4.

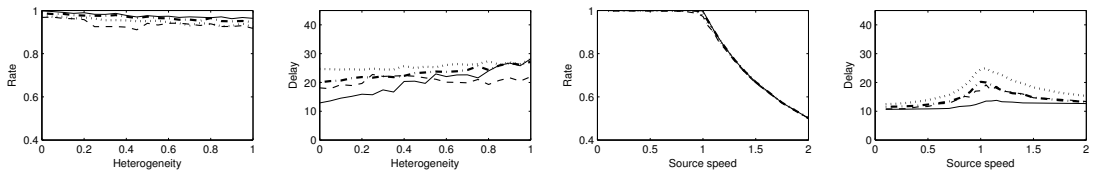
5.3 Resource aware algorithms for heterogeneous systems

The practically interesting case of heterogeneous upload capacities is much less well understood than the homogeneous case we just considered. Recall that, of all considered strategies, the *dp/ru* scheme is the only one for which optimality results exist for heterogeneous upload capacities (it is known to be rate-optimal [73]). Moreover, the considered schemes do not take into account upload capacities when performing peer selection.

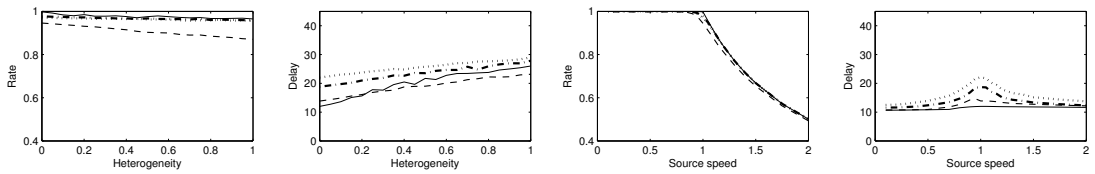
In order to investigate the impact of heterogeneity, we consider, for any fixed parameter $h \in [0, 1]$, a scenario where $u(l) = 2$ for a fraction $\frac{1}{3}h$ of the peers, $u(l) = 0.5$ for another fraction $\frac{2}{3}h$ of the peers, and $u(l) = 1$ for all other peers. The average upload capacity is therefore still equal to 1. We refer to h as the factor of *heterogeneity*: $h = 0$ corresponds to the homogeneous case, and the upload variance grows linearly with h. Figures 5.7 show the diffusion rate and



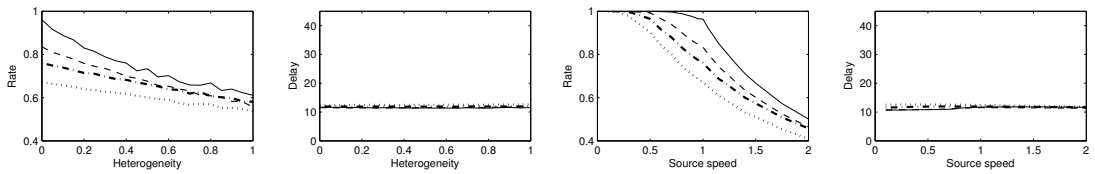
(a) dp/lu



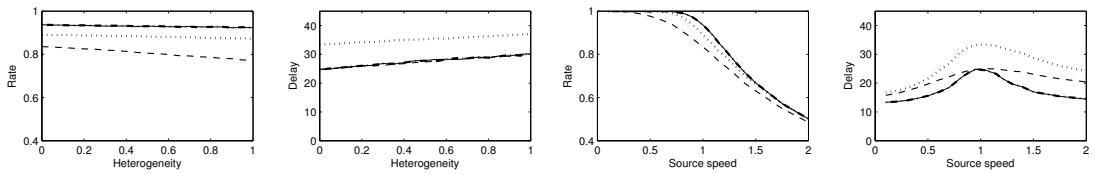
(b) lu/dp



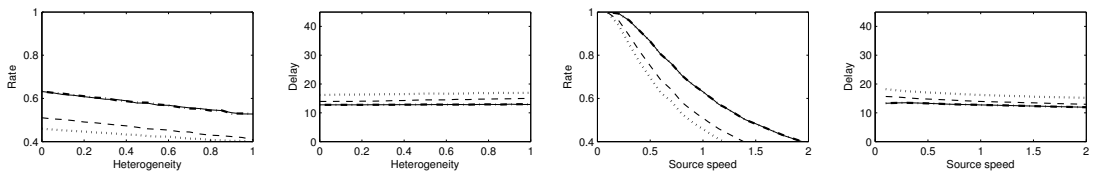
(c) lu/up



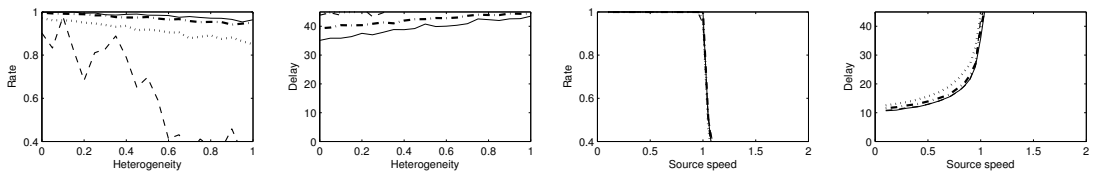
(d) lb/up



(e) rp/lu

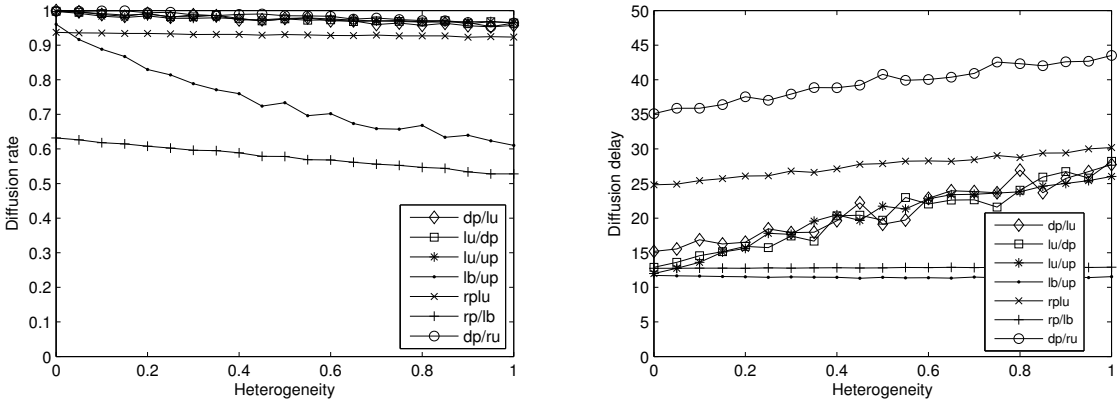


(f) rp/lb



(g) dp/ru

Figure 5.6 : Impact of restricted neighborhoods on performance.



(a) Rate (b) Delay

Figure 5.7 : Diffusion as a function of heterogeneity.

the diffusion delay of the considered schemes as a function of the heterogeneity factor in the critical regime ($\lambda = 1$).

Observe that the performance of the top three schemes (*dp/lu*, *lu/dp*, *lu/up*) worsens with h, for both rate and delay. In particular, the diffusion delay approximately doubles when h grows from 0 to 1. The impact is less significant for the *rp/lu* scheme: the diffusion rate remains approximately unchanged, while the diffusion delay increases by 25%. Regarding the *rp/lb* and *lb/up* schemes, the diffusion delay is almost insensitive to heterogeneity, but the diffusion rate is strongly impacted, especially for the latter that looses about 35%.

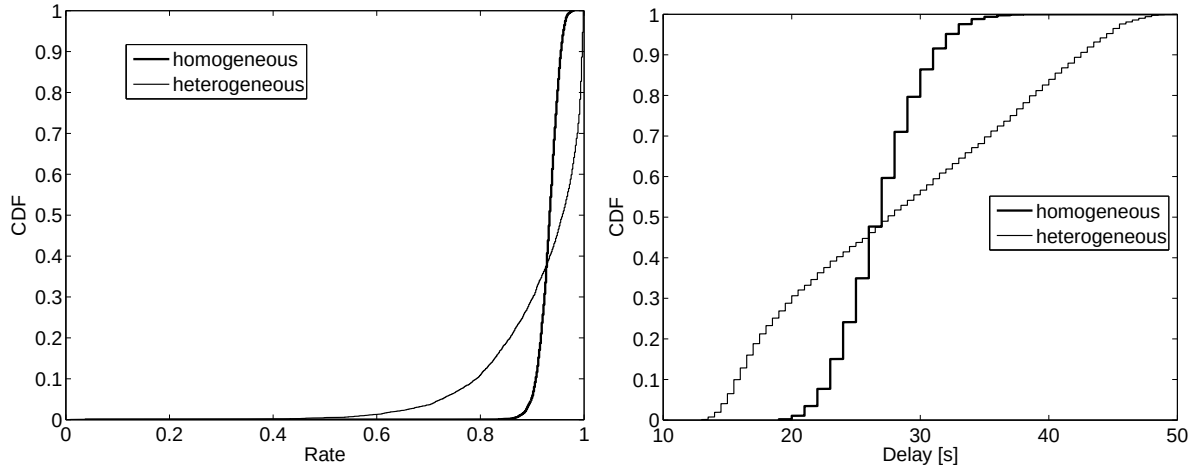


Figure 5.8 : CDF of chunk diffusion performance in case of homogeneous ($h = 0$) and heterogeneous ($h = 1$) upload capacities for the *rp/lu* scheme.

In figure 5.8 we report the CDF of chunk diffusion rate/delay for the *rp/lu* scheme in case of homogeneous ($h = 0$) and heterogeneous ($h = 1$) upload capacities. In the homogeneous case, the distributions are tightly concentrated around their averages (25 seconds for delay, and 0.93 for the rate), while in the heterogeneous case, they are scattered over a larger range of values. This indicates that a key phenomenon occurring with heterogeneity is the variability of the diffusion

of distinct chunks: while for homogeneous systems all chunks' diffusions are pretty much similar (little variance), for heterogeneous systems, some chunks are quickly disseminated with a low miss ratio while others take a longer time to achieve a lower rate.

In order to better understand this behavior, we analyze the impact the resources of the first peers receiving a given chunk have on the final diffusion performance. For a given copy number k, Figure 5.9 shows the rate/delay performance of a chunk depending on whether its k^{th} copy has been received by a rich peer ($u(l) = 2$; the thin curves) or by a poor peer ($u(l) = 0.5$; the bold curves). We observe very different diffusion rate/delay performance, especially for the earlier copies. This difference lowers with the number of chunk replicas up to the 5^{th} copy, after which the resources of the receiver do not significantly affect the final rate/delay values.

It clearly appears that the quality of a given chunk's diffusion is mostly determined by its early dissemination (where and when the very first copies of the chunk are sent): as predicted by the intuition, having the first copies of a chunk located in rich peers (in term of bandwidth) is far better than the opposite. We claim that the scattered performance distribution in the heterogeneous case is mainly due the random selection of the first chunk exchanges, which leads to different performance according to the resources of the selected peers. Because of the competition between chunks, this early differentiation can hardly be compensated after that, except if a *rarest chunk* policy is used, which is not considered here (the dispersion is reduced, but the overall performance can be impacted). For delay-aware schemes like the latest useful one, the competition actually accentuates the difference (the dissemination of under-represented chunks tends to be jammed by the dissemination of fresher, over-represented, chunks).

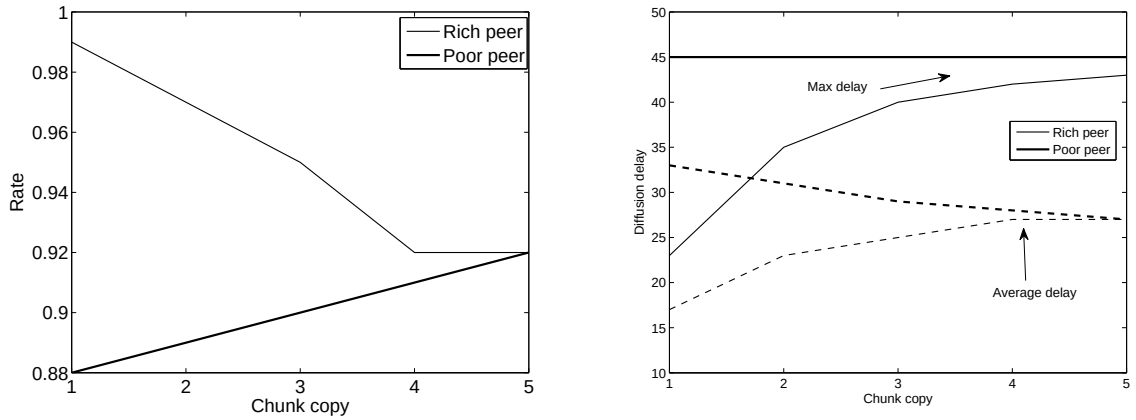


Figure 5.9 : Rate/delay performance for the *rp/lu scheme* as a function of the resources of the k^{th} peer receiving a given chunk. $h = 1$, Rich peer $u(l) = 2$, Poor peer $u(l) = 0.5$.

These results highlight the interest of using resource awareness in peer selection. In particular, the resources of the first peers receiving a given chunk are crucial for the final diffusion performance. We therefore consider diffusion algorithms that take into account the resources shared by nodes when performing the selection. As the most important resource in live streaming systems is the network bandwidth, we consider diffusion schemes targeting to be aware of the bandwidth provided by peers. Nevertheless, in the previous chapter we have highlighted that a certain level of altruism (agnostic selection) is needed for the functioning of the system. In this section, we consider this awareness-agnostic trade-off and we derive a highly versatile

model that explicitly takes this trade-off into account, and that can represent several existing resource-aware schemes, as well as new ones.

In particular, we focus on the peer selection process while for the chunk selection we just consider two simple selection policies (*latest blind (lb)* and *latest useful (lu)*) that have been shown efficient for agnostic peer selection in the previous section. We argue that to deal with heterogeneous peers, chunk selection is less crucial while it is very important to optimize peer selection. This is true only if chunks are all equal in size and if they all have the same importance. On the contrary, if some chunks have higher priority or are bigger than others, for example because they have been coded with layered techniques, the chunk selection policy plays an important role [66].

For an easier understanding of the impact the awareness has on selection policies, we consider diffusion schemes where the peer is selected first, although our model can be extended to chunk selection first. We argue that, if the chunk is selected first, the peer selection is restricted to the peers missing the given chunk, so that resource awareness is potentially limited. Consider for example, a tit-for-tat peer selection policy. If only free-riders are missing the selected chunk, the Tit-for-Tat policy has no effect on the peer selection. Moreover, peer first schemes have been shown more adapt to a practical implementation because they potentially generate low overhead while providing near-optimal rate/delay performance.

Apart for the analysis proposed in Chapter 4, mesh-based diffusion schemes designed to deal with heterogeneous upload capacities have mainly been studied by means of simulations [32, 66] or experimental evaluation [90]. Analytical studies of resource aware algorithms for P2P systems have mainly been performed for file-sharing [98, 40], or for generic applications by means of a game theory approach [17, 67, 131]. As concern live streaming, Chu et al. [27] propose a framework to evaluate the achievable download performance of receivers as a function of the altruism from the bandwidth budget perspective. They highlight that altruism has a strong impact on the performance bounds of receivers, and that a small degree of altruism brings significant benefit. In [24], the same authors propose a taxation model in which peers with more upload capacity supply the missing bandwidth of poorer peers. In [63] a game-theoretic framework is proposed to model and evaluate incentive-based strategies to stimulate user cooperation.

5.3.1 Model and algorithms

Differently from previous section, here we consider a continuous model in order to better represent the different resources shared by peers. In particular, we express the upload capacity $u(l)$ of a peer l as the amount of data per time unit it can upload. For simplicity, we assume a discrete set of U upload speeds, and classify peers in $|U|$ classes $C_1, \dots C_U$ according to their upload capacity. We denote as α_i the percentage of peers belonging to class C_i.

We suppose the stream has a constant rate of SR. The source splits it in a sequence of chunks of size c, so that a new chunk is generated every $T_{SR} = \frac{1}{\lambda} = \frac{c}{SR}$ time units. These chunks are injected into the system according to the source diffusion policy and upload constraints. The peers in turn exchange these chunks among them according to their diffusion policy, which may differ from the one of the source.

As stated before, we focus on diffusion schemes where the peer is selected first, and for the chunk selection, we just consider *latest blind (lb)* and *latest useful (lu)* policies. In both cases

(blind or useful), the sending time of peer l of class i is defined by $T_i = \frac{c}{u_i}$ if the selected chunk is indeed useful for the destination peer. If not, the destination peer can send back a notification so that the sender can select another peer.

Peer Selection Process

We now propose a general model that allows to represent various non-uniform peer selection schemes. The non-uniform selection is represented by weight functions $\{H_l\}$. A peer l associates to every neighbor $v \in N(l)$ a weight $H_l(v)$. Typical weight functions will be expressed later for some schemes. $H_i(j)$ can be time-dependent, however the time variable is implicit in order not to clutter notation.

Whenever a given peer l can upload a chunk, we assume it can use one of the two following peer selection policies:

Aware peer l selects one of its neighbors $v \in N(l)$ proportionally to its weight $H_l(v)$.

Agnostic peer l selects one of its neighbors $v \in N(l)$ uniformly at random.

The choice between the two policies is performed at random every time a chunk is sent by a peer, the aware policy been selected with a probability W, called the *awareness probability* ($0 \leq W \leq 1$). W expresses how much a peer takes resources into account when performing the selection, so that it represents the level of awareness of the diffusion scheme.

The H_l function and the W variable completely define the peer selection scheme: when a peer l can upload a chunk, the probability $\beta(l, v)$ that it selects one of its neighbors v is therefore given by

$$\beta(l,v) = \underbrace{\frac{H_l(v)}{\sum_{k \in N(l)} H_l(k)} W}_{\text{Aware}} + \underbrace{\frac{1-W}{N(l)}}_{\text{Agnostic}} \tag{5.7}$$

In the following we express H and/or W for some peer selection schemes. Remember that in this section we focus on diffusion schemes where the peer is selected first. This means that, unless otherwise specified, a sender peer has no prior knowledge about the buffer state of its neighbors, so it is not guaranteed that it will have useful chunks for the peer it will select.

Random peer selection (*rp*)

The random peer selection it the limit case where peers are completely unaware of their neighbors' characteristics as deeply analyzed in the previous section. We then have $W = 0$, and there is not need to define a weight function. This results in

$$\beta(l,v) = \frac{1}{N(l)}.$$

Bandwidth-aware peer selection (*ba*)

This is the simplest scheme taking into account the resources nodes devote to the system. A peer l selects one of its neighbors $v \in N(l)$ proportionally to its upload capacity, so we have $H_l(v) = u(v)$. Note that in the homogeneous upload capacity case, the selection is equivalent to the uniformly random selection.

This scheme has been introduced by da Silva *et al.* in [32]. However there are two main differences between our model and the framework they propose: in [32],

- the chunk is selected first, and the bandwidth-aware selection is performed among the neighbors that need the selected chunk from the sender.
- the selection scheme is fully-aware (corresponding to $W = 1$ in our model), while we propose to discuss later the influence of the awareness probability W.

Although we focus on a edge-constraint scenario, the upload estimation may differ in practice depending on the measurement points. Our model could be easily generalized by setting $H_l(v) = u_l(v)$, where $u_l(v)$ is the available bandwidth capacity from v to l.

Tit-for-Tat peer selection (tft)

Tit-for-tat mechanisms have been introduced in P2P by the BitTorrent protocol [30], and have been widely studied for file sharing systems. In the previous section, we have shown these mechanisms are also very effective in live streaming applications.

In the original BitTorrent protocol, a subset of potential receivers is periodically selected [30]. Following the authors in [66], we propose a simpler protocol where a receiver peer is selected every time a chunk is sent. We propose to drive the peer selection by using as weight function $H_l(v)$ an historic variable that is computed every *epoch* T_e; this historic value indicates the amount of data peer l downloaded from peer v during the last epoch. In this way, a peer v is selected by a peer l proportionally to the amount of data it provided to l during last epoch.

Data-driven peer selection

The model we introduced so far is not only able to describe the behavior of resource aware algorithms, but also to represent diffusion schemes that take into account the collection of chunks B when performing peer selection.

The *most deprived selection* presented in 5.2.1 , as well as the *proportional deprived selection* proposed by Chatzidrossos et al. [21], can be represented by our model.

The former selects the destination peer uniformly at random among those neighbors v of l for which $|B(l) \setminus B(v)|$ is maximum. The weight function can be expressed as:

$$H_l(v) = \begin{cases} 1 \text{ if } |B(l) \setminus B(v)| = \max_{v \in N(l)} |B(l) \setminus B(v)|, \\ 0 \text{ otherwise.} \end{cases} \tag{5.8}$$

The latter selects a destination peer v proportionally to the number of useful chunks the sender peer l has for it. The weight function can be expressed as $H_l(v) = |B(l) \setminus B(v)|$.

In the following we are not going to analyze these data-driven peer selection schemes because we focus on resource-aware policies. However, the recursive formulas derived in Section 5.3.2 are also valid for data-driven peer selection policies.

Implementation issues

The simplicity and strength of the bandwidth-aware selection comes from the fact that it directly uses the amount of bandwidth provided by a node as weight function. The upload capacity can be measured by means of bandwidth estimation tools, or can be provided by an external oracle/tracker. However, both approaches highlight several practical drawbacks.

In the case of measurements made by the peers themselves, known bandwidth estimation tools may be inaccurate, particularly when used in large-scale distributed systems [31]. Moreover, the measured value may vary over time according to network condition, so that the measurement should be frequently repeated generating high overhead and interference.

If some tracker or oracle is used, the upload capacity monitored by the central authority can be a nominal one, provided by the peers, or can be inferred from measurements made from different points. Apart from accuracy issues, the authority providing the information, as well as the measurement points, should be trusted and should not cheat on the values it provides.

In our model we do not take all these issues into account, but we argue that this scheme is currently hard to implement in real systems. However, some projects, like Napa-Wine [79], or standardization efforts, like ALTO [120], are working in order to provide reliable resource-monitoring to peers by using both oracle and measurements at nodes.

On the other hand, with *tit-for-tat* mechanisms every peer can easily evaluate the amount of data provided by its neighbors. This information is trusted and very accurate while it requires no overhead at all. Moreover, we have shown in the previous chapter that *tit-for-tat* mechanisms are efficient to improve the system performance, and they are able to discriminate peer resources, giving advantages to nodes contributing the more to the system.

As concern data-driven peer selection, it is known to provide optimal performance for specific scenarios [73], but it generates a lot of overhead and suffers of strong performance degradation if the neighborhood is restricted. Moreover, this selection scheme is very sensitive to *cheating* because it is based on information provided by neighbors. In fact, a peer can largely increase the probability of being selected by simply advertising altered chunk collections and pretending to possess less chunks than it actually does.

5.3.2 Recursive formulas

In this section we derive recursive formulas for a generic diffusion scheme based on an *aware peer* selection coupled with a *latest blind chunk* selection. The *latest useful* selection will be the subject of the next simulative section.

In case of heterogeneous upload capacities the rate/delay distribution is not centered around a given value but scattered over a large range. In order to represent the diffusion functions in such a scenario, it is therefore more relevant to work with distribution instead of using only averaged values (which suffices in the homogeneous case analyzed in the previous section).

As the performance is mainly affected by the first chunk exchanges, we propose a two-steps approach: first an exact description of the early behavior of the diffusion, then the use of averaged approximation to derive the rest of the diffusion process.

Let J be a distribution of system states that describes the early behavior of a chunk's diffusion. One may think of J as the *initial conditions* of the diffusion. These initial conditions represent different possible evolutions of first chunk exchanges up to a certain time T_{init}. We propose to use J to compute a recursive approximation of the afterwards diffusion. The larger (and the more accurate) the initial conditions are, the better the distribution computed by the recursive formulas will fit the real distribution.

The initial conditions should be deterministically computed according to the diffusion scheme (see below); such operation can be computationally expensive and exponentially time consuming (we have to limit ourselves to the early diffusion). However, the number of exchanges to compute is reasonable in practice: for instance, for a population of 600 peers, we have observed that the exact exchanges that occur after 5 chunk copies do not significantly impact the final diffusion performance. This keeps the recursive approach mush less expensive in term of computational resources and time with respect to a complete simulative analysis.

We assume a scenario where every peer has a complete knowledge of the overlay (full mesh connectivity) and that the H and W parameters are the same for all peers. We also suppose that the resources shared by a node are defined by its class, so we can express the probability that a peer of class i' is selected simply as $\beta(i')$.

As for the recursive formulas derived in Section 5.2.2 we assume that the number of peer is sufficiently large, so that the system may be considered in the mean field regime where peers are mutually independent, and that the probability that a given chunk belongs to $B(l)$ is independent from the fact that any other chunk belongs to $B(l)$.

We make the approximation that all peers of the same class are synchronized in uploading a chunk. 0 being the time of one given chunk's creation, we define $\mathcal{T}_i := \{T_i\ 2T_i\ 3T_i\ ..\}$ as the set of times at which peers of class i may send a chunk, and $\mathcal{T}_{SR} := \{T_{SR}\ 2T_{SR}\ 3T_{SR}\ ..\}$ as the set of chunk generation times. We define $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup ... \cup \mathcal{T}_U \cup \mathcal{T}_{SR}$ as the (sorted) set of times at which an event occurs. Simultaneous events from distinct classes are taken into account with their multiplicity.

The values we are interested in are the fraction of the peers of a class that received the chunk before time t. For every instant of time $t \in \mathcal{T}$ and each class i, we propose to compute that fraction, denoted as $r_i(t)$.

The first step is to compute the initial conditions J. A set of $|J|$ instances of the $r_i(T_{init})$ are generated according to the considered scheme. Note that for an instance $j \in J$, all $r_i(T_{init})$ are deterministic. Starting from these initial conditions the recursive formulas describe the diffusion function for each $j \in J$. In the following when considering a given $r_i(t)$, we assume implicitly an initial condition $j \in J$, while the average over J is denoted as $\overline{r}_i(t)$.

For every time $t \in \mathcal{T} : t > T_{init}$ at which an upload event occurs, we denote as i the class sending the chunk at that time t, and as t' the instant of time preceding t in $\mathcal{T}$. We denote as $p(t)$ the probability that a given peer ends the upload of the chunk at time t, so that on average $np(t)$ transmissions of the considered chunk finish at time t. $p(t)$ is initially set to 0 for all t values. That probability $p(t)$ is spread over the U classes according to the selection probability β, so that peers in class k receive the tagged chunk at time t with probability $\alpha_k\beta(k)p(t)$. Among a

given class target peers are then selected uniformly at random. Due to this random selection, the number of copies of the tagged chunk that are received by an arbitrary peer is a binomial random variable with parameter $(\alpha_k n, \beta(k)p(t)/\alpha_k n)$. For large n, this can be approximated by a Poisson random variable with mean $\beta(k)p(t)$. The probability that a peer of class k receives at least one copy of the tagged chunk at time t is therefore approximately equals to $1 - e^{-\beta(k)p(t)}$. A fraction $1 - r_k(t)$ of the peers that receive the chunk at time t actually need it. The recursive formula is then:

$$\forall k : 1 \leq k \leq U, r_k(t) = r_k(t') + (1 - e^{-\beta(k)p(t)})(1 - r_k(t')) \tag{5.9}$$

We then need to update the value of $p(t)$ for the later event in $\mathcal{T}_i$. This means to compute the probability that the chunk is the latest in the collection of chunks B of peers of class i. This affects the probability that the download of the tagged chunk ends at time $t + T_i$ as follow:

$$p(t + T_i) = p(t + T_i) + \alpha_i r_i(t) \prod_{k=1}^{\lfloor \frac{t}{T_{SR}} \rfloor} (1 - \overline{r}_i(kT_{SR})) \tag{5.10}$$

For every time $t \in \mathcal{T}_{SR} : t > T_{init}$, at which a new chunk is generated, the status of the considered chunk is unchanged (no transmissions occur for it) so we simply have:

$$\forall k : 1 \leq k \leq U, r_k(t) = r_k(t') \tag{5.11}$$

Formula validation

We validate the recursive formulas by considering the *ba* peer selection process with awareness probability $W = 1$. We suppose the overlay is a complete graph and the source injects only one copy of each chunk in the system ($T_{SR} = T_S$). To this goal we set the chunk size to $c = 0.9\ Mb$ and the source upload capacity to $u_S = 0.9\ Mbps$. The other parameters are those of the reference scenario described in the next section.

We consider two different sets of initial conditions: J_1 and J_2. The former is composed of only one initial condition ($|J_1| = 1$), and it is only based on the copy uploaded by the source ($T_{init} = T_{SR}$). In this case, we will only have one rate/delay value and not a distribution. The latter is composed of $|J_2| = 1000$ different initial conditions, and is based on $T_{init} = T_{SR} + 1\ s$ (given the system parameters used, an initial condition represents 5 chunk exchanges on average). In this case, we will have a distribution based on 1000 different chunk diffusions.

Figure 5.10 shows formulas are quite accurate in predicting the rate/delay performance of the considered scheme. As expected, to increase the number of initial conditions and T_{init}, increases the accuracy of the performance prediction. In particular, the distribution based on 1000 samples of 5 chunk exchanges fits pretty well the distribution based on a simulation of 10000 chunks. It is possible to observe estimation errors between 0-7% (C4-C2) as concern diffusion rate, and 10-15% (C1-C4) as concern the average delay.

These errors are slightly larger than in the homogeneous case studied in 5.2.3. This is due to the variability of the diffusion process that is more stressed in heterogeneous systems because of the additional randomness of the different upload capacities. Nevertheless the obtained results are worthwhile for having a fast performance estimate of a system.

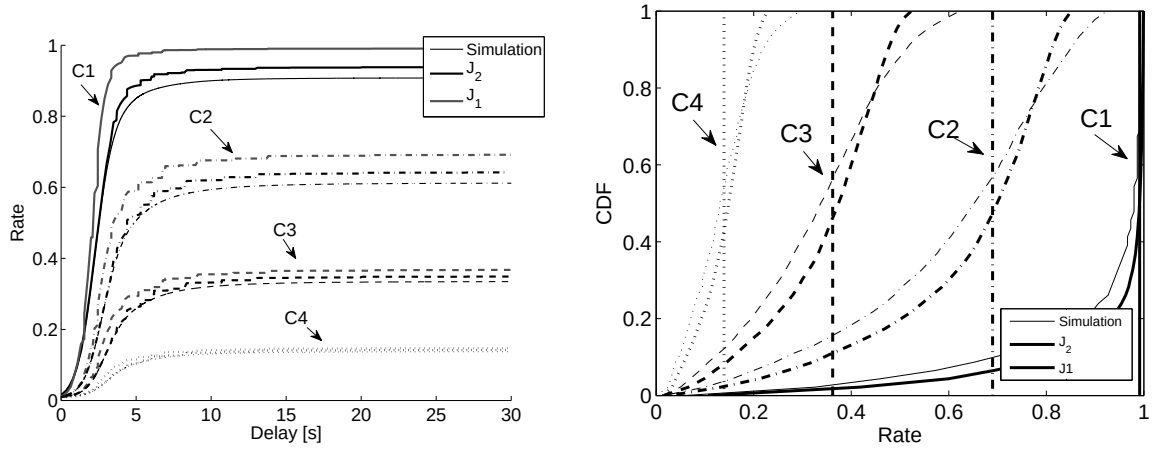


Figure 5.10 : Per class validation of the recursive formulas. *ba* peer selection.

5.3.3 Simulation results

In this section, we evaluate the rate (or miss ratio)/delay[4] trade-off achieved by resource aware selection schemes. In particular, we focus on the performance of three representative peer selection policies: *random peer* (*rp*), *bandwidth-aware* (*ba*) and *tit-for-tat* (*tft*).

To this purpose we use a customized version of an event-based simulator developed by the Telecommunication Networks Group of Politecnico di Torino[5] where we implement the aforementioned schemes.

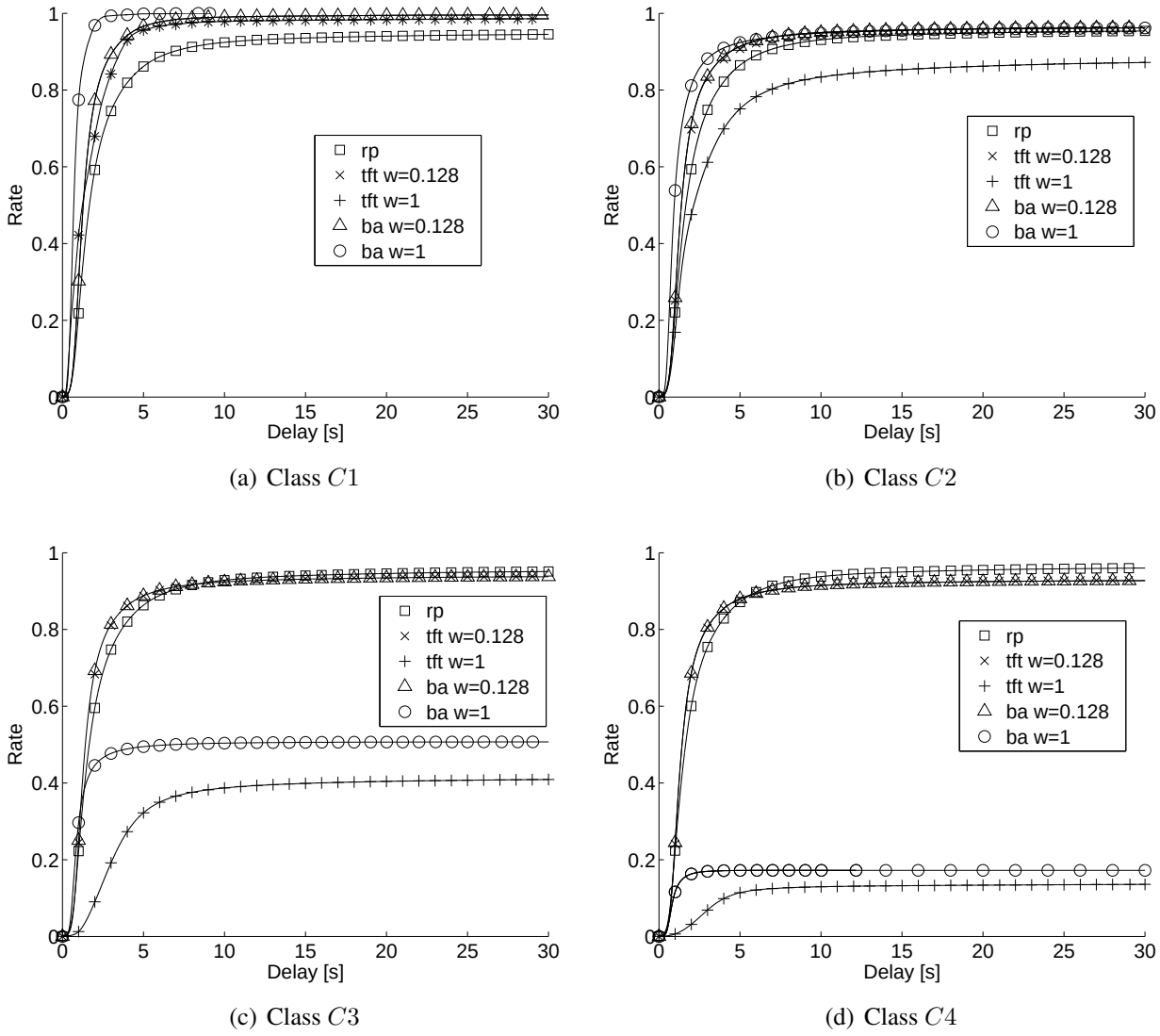
Unless otherwise stated, we suppose there are $n = 1000$ peers and we set their uplink capacities according to the distribution reported in Table 5.2, that is derived from the measurement study presented in [8], and that has been used for the analysis in [47]. We suppose every peer has about 50 neighbors, $N(l) \approx 50$[6]. The source has about 50 neighbors as well, an upload capacity $u_s = 1.1\ Mbps$ and employs a *rp* selection policy.

In order to avoid critical regime effects, we suppose the stream rate $SR = 0.9\ Mbps$ that leads to a bandwidth balance of $1.13\ SR$. We set the chunk size $c = 0.09\ Mb$, we suppose peers have a buffer of 30 seconds and for the *tft* scheme the epoch length is set to $T_e = 10\ s$.

The chunk selection policy we consider here is *latest useful*.

Class	Uplink [Mbps]	Percentage of peers
C1	4	15%
C2	1	25%
C3	0.384	40%
C4	0.128	20%

Table 5.2 : Upload capacity distribution with mean 1.02 Mbps.



(a) Class $C1$ (b) Class $C2$

(c) Class $C3$ (d) Class $C4$

Figure 5.11 : Chunk diffusion in the reference scenario

Reference scenario

We first consider a reference scenario whose diffusion process of the different schemes is pictorially represented in figure 5.11 for all classes. For *ba* and *tft* peer selection we consider two values of awareness probability: $W = 1$ and $W = 0.128$ corresponding to a fully-aware and a generous approach respectively.

We observe schemes taking into account peer contributions/resources tend to decrease the diffusion delay with respect to the agnostic *rp* for all classes. *ba* gives priority to richer peers, so that the diffusion process is speeded up thanks to their high upload capacity placed at the top of chunk diffusion trees. On the other hand, *tft* clusters peer according to their resources [40], leading to a similar effect as observed in the experimental analysis presented in the previous

[4]We consider here the average delay that is the time needed for a chunk to reach a peer on average.
[5]`http://www.napa-wine.eu/cgi-bin/twiki/view/Public/P2PTVSim`
[6]We consider G is an Erdös-Renyi graph with edge probability equal to 0.05.

chapter.

Such resource aware schemes increase the diffusion rate of the richer classes C1-C2, while they reduce the one of poorer classes C3-C4. This rate decrease is particularly dramatic in case of a completely aware selection (W=1). On the other hand, if the selection is more generous (W=0.128), this drastic reduction is avoided, but the diffusion delay increases with respect to a fully-aware approach.

This clearly highlights a rate/delay trade-off as a function of the awareness probability W.

Awareness-Agnostic peer selection trade-off

Figure 5.12 reports the rate/delay performance of *ba* and *tft* schemes as a function of the awareness probability in the heterogeneous scenario described in Table 5.2.

The diffusion delay decreases as the awareness probability increases for all bandwidth classes. This indicates the placement of the nodes with higher upload capacities at the top of the diffusion trees effectively speeds up the diffusion process. We also notice that, by increasing the awareness probability, the delay gap between different classes increases as well. In particular, when $W \approx 0$, all classes achieve the same diffusion delay because the selection is almost random (as in *rp*). On the other hand, when $W = 1$, the discrimination is maximal because the selection is purely aware. In fact, more and more peers with higher upload capacities are selected first as the awareness probability increases.

As concern miss ratio, richer classes take advantage of the increasing awareness. On the other hand, the miss ratio of the poorer classes stagnates until a certain awareness value of about $W = 0.22$, after which peers start missing more and more chunks. The intuition is that richer peers are selected with increasing frequency decreasing their miss ratio and, as a consequence, poorer classes tend to be "forgotten".

We observe *ba* scheme slightly outperforms *tft*. This is not surprising: *ba* weights peers according to their upload capacity, so that it perfectly discriminates them according to their resources. However, the gap is very small making *tft* appealing for real deployment because more simple and reliable than *ba*.

Notice that a pure *tft* approach ($W = 1$) performs poorly: without agnostic disseminations, the peer clustering generated by *tft* interferes with the chunk dissemination. This does not happen under *ba* scheme because every peer can be selected with low probability, even poorer ones, giving a minimal chance for a chunk to reach poorer peers.

Class	Uplink [Mbps]	Percentage of peers
$\tilde{C}1$	3.5	7%
$\tilde{C}2$	0.35	66%
$\tilde{C}3$	0.2	27%

Table 5.3 : Upload capacity distribution with mean 0.53 Mbps.

In order to validate our claims, we consider another scenario (Table 5.3) which is derived from the measurement study presented in [29], and it has been used for the evaluation in [32]. We also consider the case of free-riders by setting the upload capacity of peers of class $\tilde{C}3$ to 0. In order to have the same bandwidth balance as in the previous scenario, we reduce the stream rate

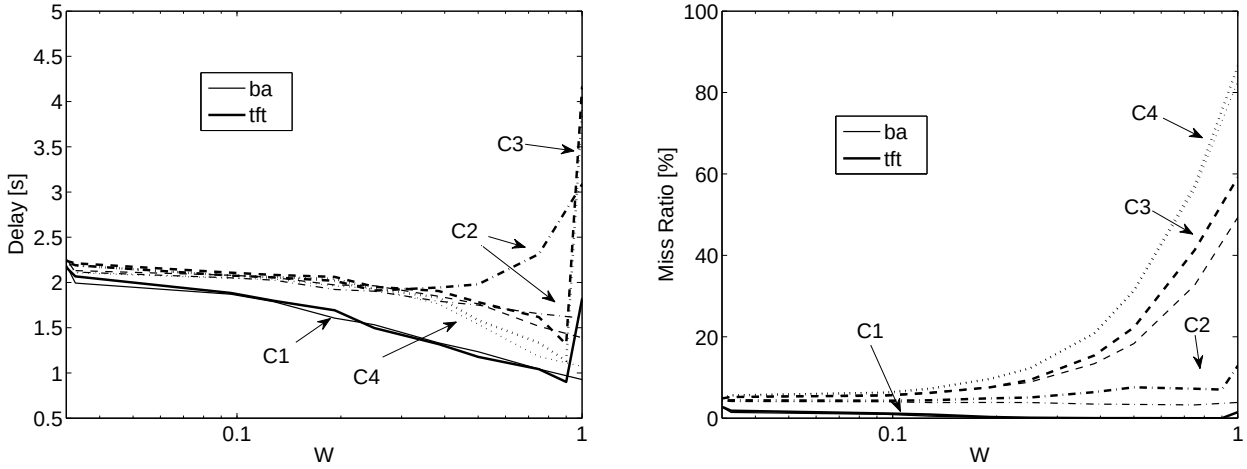


Figure 5.12 : Diffusion delay and chunk miss ratio as a function of the awareness probability.

to $SR = 0.5\ Mbps$ and the chunk size to $c = 0.05\ Mb$. Note that in this scenario the bandwidth distribution is more skewed. Since the two selection policies behave similarly, in the following we focus on *tft* peer selection.

Figure 5.13 highlights the trend in the *3 classes* scenario is similar to the one observed before. The only difference is that the gain of the increasing awareness is more evident for all classes. This is due to the high bandwidth of the first class with respect to the stream rate: as soon as this class is privileged all peers improve their performance.

In the scenario with *free-riders*, all chunks the source uploads to class $\tilde{C}3$ are lost because peers cannot upload them. So the miss ratio cannot be lower than the percentage of peers of class $\tilde{C}3$. Classes $\tilde{C}1$ and $\tilde{C}2$ almost receive all the other chunks while free-riders are identified and receive a decreasing percentage of data as the awareness probability increases. This highlights that, in an heterogeneous scenario, the selection policy employed by the source can have a tremendous impact on the system performance. If the source could discriminate peers according to their resources, we won't observe such a miss ratio. We better investigate in the following the impact of different source selection schemes.

In all scenarios we observe the presence of a minimum suitable value of awareness probability. Empirically, it does not seem interesting to select an awareness probability $W < 0.1$ because there is almost no gain with respect to the *rp* selection. From this value to $W = 1$ ($W = 1 - \epsilon$ for *tft* scheme) a trade-off arises. The more the scheme is aware the more richer peers improve their performance. On the other hand, even if there is enough bandwidth, peers of the poorer classes loose lot of chunks. This can be seen as a good property of the system because it incentives peers to contribute more to the system in order to improve their performance. On the other hand, part of the bandwidth is lost. The best value for the awareness probability depends on the application environment but in any case this value should be larger than 0.1 in order to discriminate peers according to their resources, to improve system performance and to recompense peers contributing the more.

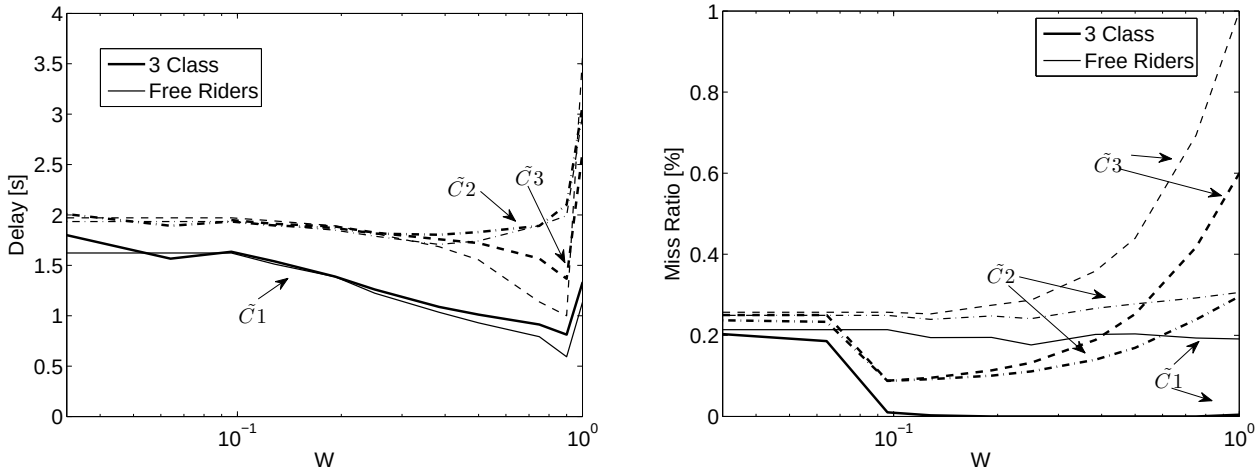


Figure 5.13 : *tft* performance as a function of awareness parameter for a skewed bandwidth distribution and in presence of free-riders.

Source scheduling

We now analyze the impact of the source selection policy and of the source upload capacity on the scheme diffusion performance.

In Figure 5.14, we consider four different source policies: random peer selection (*rp*) with source upload capacity $u_s = SR$; random peer selection with source upload capacity $u_s = 4\,SR$; selection of a peer of class $C1$ with upload capacity $u_s = SR$; selection of a peer of class $C4$ with upload capacity $u_s = SR$. Since the trend of all classes is similar we only report in figure the performance of peers of class $C1$.

The diffusion delay strongly depends on the source policy. In fact, the selection of a peer of class $C1$ can reduce of 3 times the delay with respect to the selection of a peer of class $C4$ while the *rp* selection stays in between. As explained earlier, it is very difficult to estimate the upload capacity of peers, and the source cannot employ a *tft* mechanism because it does not download any data. It is interesting to observe that, if the source has an upload capacity of $u_s = 4\,SR$, the *rp* selection performs as the selection of a peer of class $C1$. This means that, if the source is "slightly" over-provisioned (remind that the system should handle thousands of peers, so a source bandwidth of a few SR can be considered negligible with respect to overall bandwidth involved) it does not need to discriminate peers according to their resources.

As concern miss ratio, we observe a dramatic degradation if the source sends the first copy of every chunk to a peer of class $C4$. These peers have not enough capacity to distribute enough copies before new chunks are injected into the system inhibiting the diffusion of the old chunk. All the other policies can provide similar miss ratios.

We now deeper investigate the impact of the source upload capacity when it performs *rp* selection. Results are reported in Figure 5.15 for $C1$ and $C4$ in case of *rp* and *tft* selection at nodes.

The diffusion delay decreases as the number of copies of each chunk injected by the source increases. The decrease is particularly significant for the first additional copies ($u_s = 2 - 3 -$

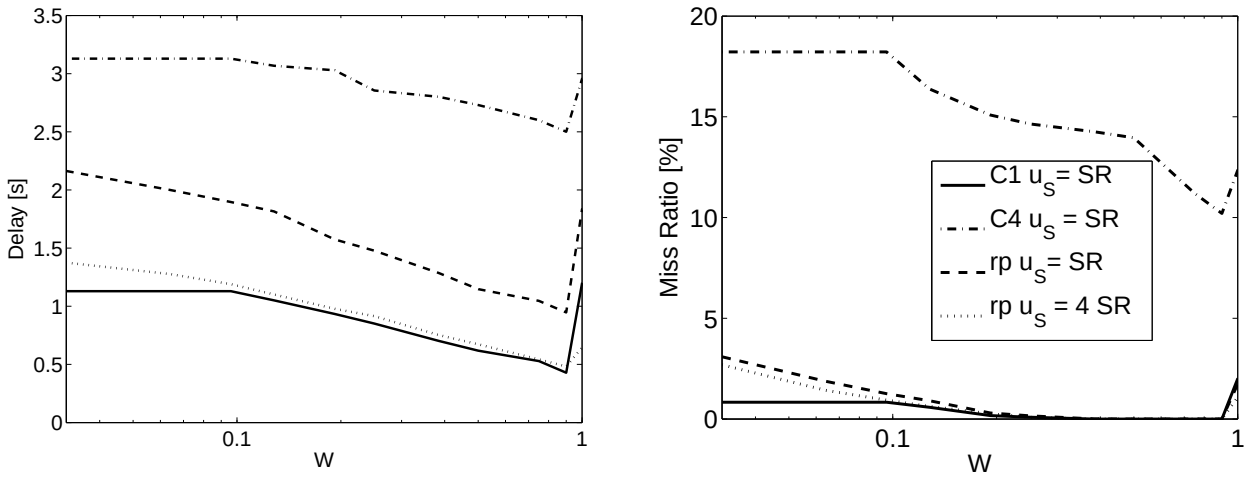


Figure 5.14 : Diffusion delay and miss ratio of $C1$ peers as a function of awareness probability for different source selection polities. *tft* selection at nodes.

$4\ SR$): more peers receive the fresher chunk and start its diffusion being *latest useful* the chunk selection policy. As concern miss ratio we observe almost no gain by increasing the source capacity.

The variance of both delay/miss ratio decreases by increasing the source upload capacity. Again, the first additional copies highlight the larger variance decrease. This indicates the chunk diffusion is more stable, and schemes can provide similar performance for the different chunks by increasing the source upload capacity.

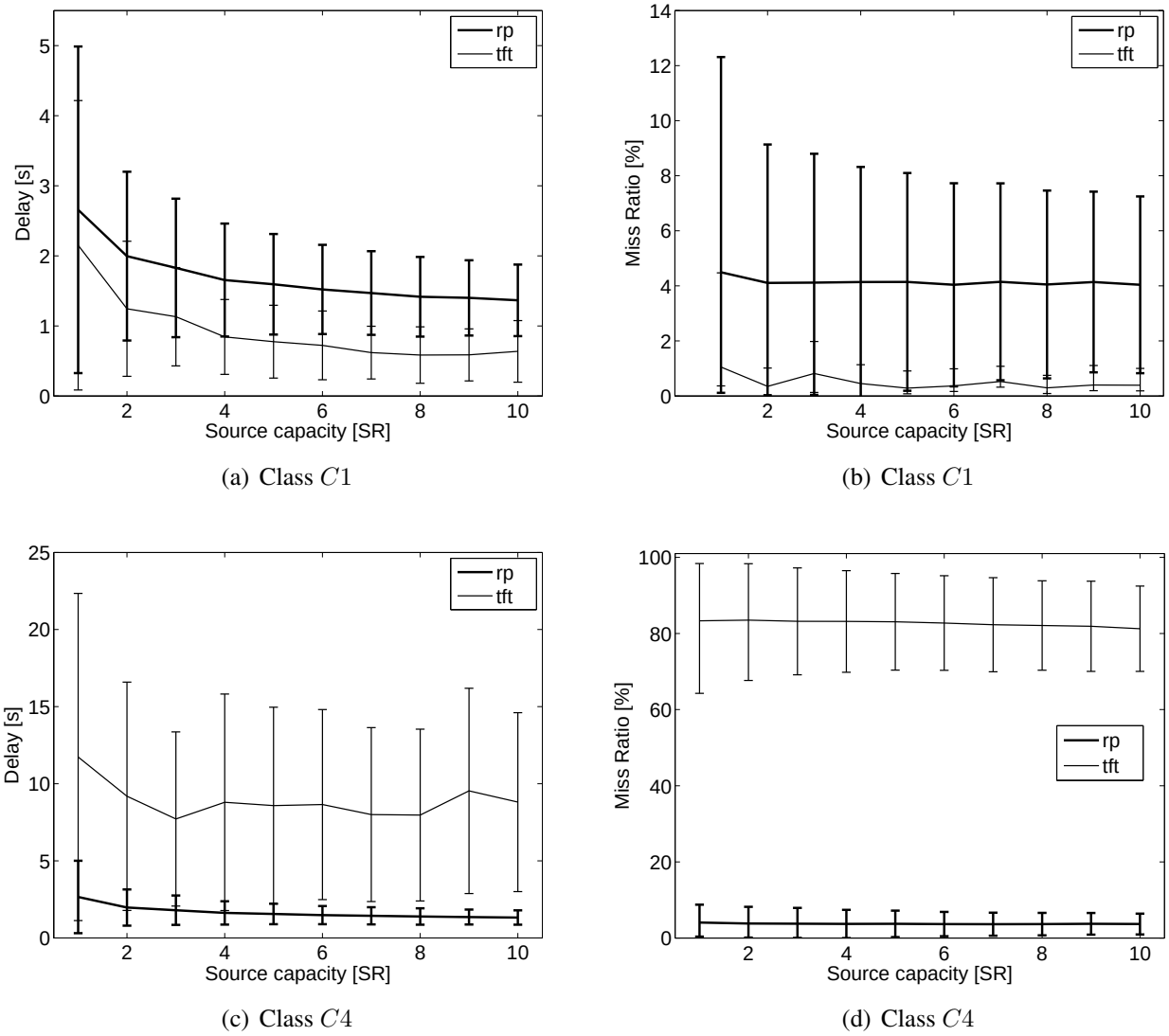
Convergence time and epoch length

So far, we have highlighted that *tft* behaves similarly to *ba* peer selection while being more appealing for real deployment. Such a scheme is driven by the evaluation of peer contributions performed every epoch T_e. As a consequence, algorithms based on *tft* may need a certain period of time, called *convergence time*, before they reach a steady-state where their performance (diffusion delay and miss ratio) is stable.

The convergence properties of *tft* have already been analyzed for file-sharing applications in [40]. We investigate in this section the convergence time of *tft* peer selection in live streaming systems, and we evaluate the impact the epoch length T_e has on their performance. In a live streaming system the *convergence time* indicates the time needed to reach both stable diffusion delay and miss ratio.

Figure 5.16 indicates the diffusion delay decreases as the epoch length increases for all bandwidth classes. The miss ratio decreases as well only for richer classes, while for the poorer classes it stagnates or slightly increases. The larger evaluation time allows peers to better estimate the resources provided by their neighbors. As a consequence, the peer selection is more accurate and all peers improve their performance with respect to a *rp* selection.

The price to pay is that longer epoch times require longer convergence times as showed in Figure 5.17. In details, peers of the richer classes require more time to reach a stable performance



(a) Class $C1$

(b) Class $C1$

(c) Class $C4$

(d) Class $C4$

Figure 5.15 : Diffusion delay and miss ratio (average value and variance) as a function of the source upload capacity. Random peer selection at source.

for small awareness parameters or short epoch lengths. Under such values only peers of the richer classes have performance different from *rp* selection, so they are the only to need time to stabilize their performance. On the contrary, when W or T_e increases, the convergence time of poorer classes strongly increases. In such a case, the performance of the poorer classes is also affected, and, as a consequence, their convergence time increases and is eventually longer than the one of the richer classes.

5.4 Optimizing parameters

So far we have focused on the design and the analysis of chunk exchange algorithms and we have highlighted a good scheme is essential to mesh-based live streaming systems. For a given scheme however, an optimization at a detailed level is also important. This involves the fine

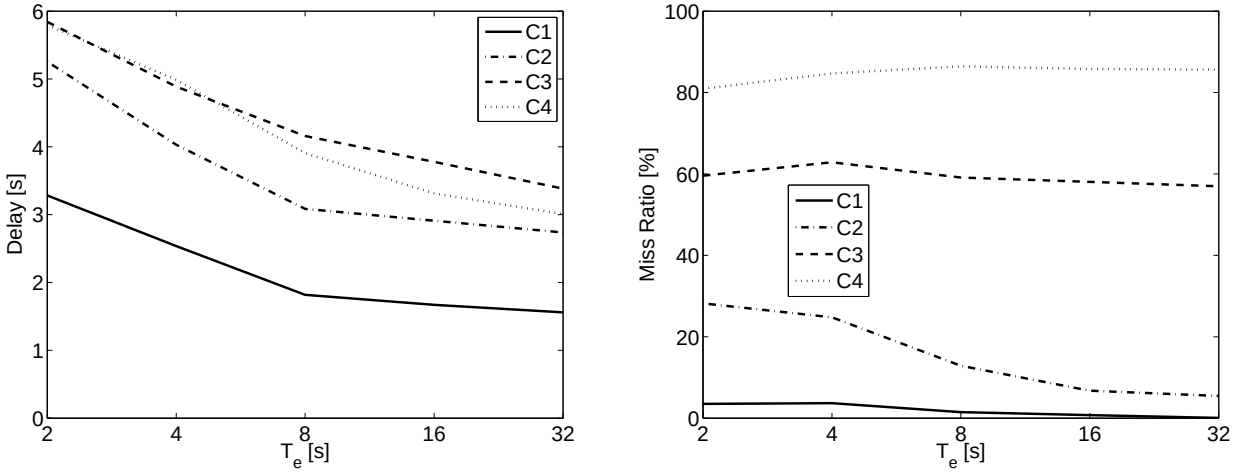


Figure 5.16 : Diffusion delay and miss ratio as a function of the epoch length T_e.

tuning of dissemination parameters, such as chunk size, receiver buffer size, number of peers to probe, etc. Intuitively, the chunk size has a significant impact on performance, since smaller chunk sizes may be more efficient but incur relatively higher overhead, and larger chunk sizes have lower overhead but may result in higher delay. The receiver buffer size (relative to chunk size) impacts the diversity in choice available to a peer for transmission. In the scheme with random peer choice, probing more than one peer for the decision of chunk exchange may help (power of choices), but it also increases overhead. These are some of the finer details of any dissemination scheme that must be closely examined.

There has been some study on parameter sizing for peer-to-peer file sharing systems. In [70] it is shown that small chunk sizes are not always best for file transfer; [58] proposes uplink allocation strategies designed to improve uplink utilization of BitTorrent-like systems. However, results obtained for file sharing systems are not directly applicable to live streaming applications. First, a newly created chunk should be disseminated as fast as possible in live streaming, so there is a strong delay component, naturally limiting the chunk size. Secondly, missing chunks may be acceptable if a resilient codec is used, so optimal values are not always comparable to those in the file transfer case. Then, the buffer size, which is a parameter specific to streaming, can impact the performance (see for instance [132]).

In this section, we investigate dissemination parameters in mesh-based peer-to-peer live streaming through extensive simulations. In particular, we focus on the chunk size, on the probe set size and on the number of parallel upload connections of dissemination algorithms where the peer is selected first.

5.4.1 Metodology

For our analysis we use the event-based simulator already used in the previous section that we modify to take network latencies, control overhead and parallel upload connections into account.

With respect to the model used in the previous section, here we assume that every link con-

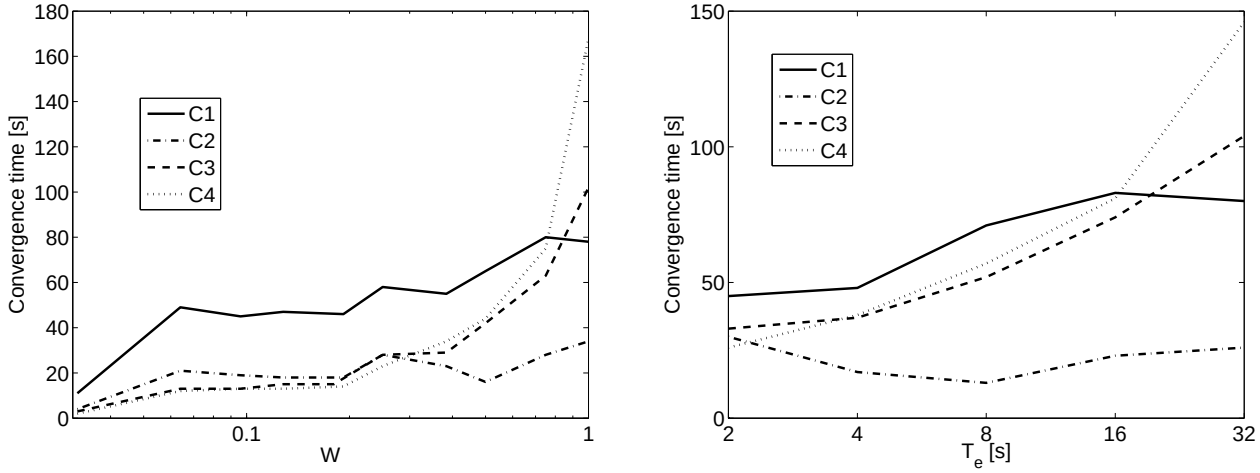


Figure 5.17 : Convergence time as a function of the awareness probability W for $T_e = 10\ s$, and of the epoch length T_e for $W = 0.75$.

necting a pair of peers $\{l, v\}$ is characterized by a constant round trip delay RTT_{lv} and is lossless. We further assume that there are no queuing nor processing delays, so the *transfer delay* (the time for a chunk or control packet to travel from peer l to peer v) is equal to $transmission\ delay + \frac{RTT_{lv}}{2}$. The choice of such a network model allows us to obtain results that are not affected by transport network congestion or losses.

We consider three representative diffusion schemes where the peer is selected first: *random peer / latest blind chunk* (*rp/lb*), *random peer / latest useful chunk* (*rp/lu*) and *bandwidth aware peer / latest useful chunk (*ba/lu*)*.

Every peer periodically selects a subset m' of its neighbors, according to one of the aforementioned algorithms (that is random or bandwidth-aware selection), and probes them in order to discover their missing chunks, except for the case of the *latest blind* scheme. We refer to the set of neighbors probed as the probe set. Based on the responses possibly received, the peer then transmits corresponding chunks.

A peer can upload a chunk to at maximum m peers in parallel by fairly sharing its upload bandwidth. It may happen that a peer cannot serve m recipients because it does not have enough useful chunks. In that case it uploads the chunks faster (since there are less than m active connections), but it may stay idle for the subsequent period of time (because it needs to acquire new chunk maps from newly selected peers). An additional overhead is taken into account at every peer to reply to control messages coming from potential sender peers.

Unless otherwise stated we consider a network of $n = 1000$ peers, all with the same upload bandwidth $u_l = 1.03Mb/s$, an unlimited download bandwidth and about 50 neighbors, $N(l) \approx 50$[7]. We set the stream rate $SR = 0.9Mb/s$. Latencies between nodes are taken from the data set of the Meridian project [97]. A buffer of size up to 300 chunks is available at all peers, in order to avoid possible missing chunks due to buffer shortage (this implies a buffer size proportional to the chunk size).

[7]We consider G is an Erdös-Renyi graph with edge probability equal to 0.05.

5.4.2 Chunk size and performance

As a first experiment, we analyze the chunk miss ratio as a function of the chunk size. The results are shown in Figures 5.18 to 5.20, for the *rp/lu* scheme with $m = m'$ varying from 1 to 5.

Chunk miss ratio

In Figure 5.18, we observe two cases:

- For large chunks (in our experiment, c greater than a few hundred kilobits, the exact value depends on the number of simultaneous connections m), there are no missing chunks.
- As the chunk size goes below a certain critical value, chunks start to miss, roughly proportional to the logarithm of the chunk size.

This phenomenon can be explained as follows: the time between two consecutive chunks is $\frac{c}{SR}$, and is therefore proportional to the chunk size c. When c is big enough (all other parameters being the same), we can assume that more and more control messages per chunk can be exchanged between peers. This should achieve a proper diffusion, provided enough bandwidth is available, since a sender peer will have enough time to find a neighbor needing a given chunk. On the contrary, when $\frac{c}{SR}$ is too small, peers do not have enough time to exchange control messages, resulting in missing chunks. Note that increasing m slightly improves the performance.

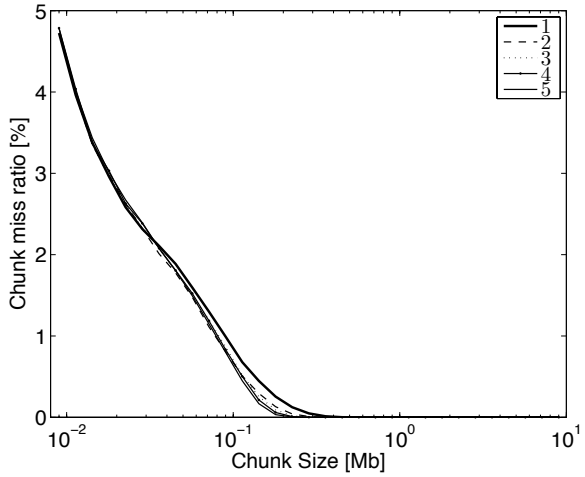


Figure 5.18 : Chunk miss ratio as a function of the chunk size. $m = m'$ varying from 1 to 5.

Delay

The average diffusion delay as a function of the chunk size is shown in Figure 5.19. The main result is that the delay is proportional to the chunk size[8], and grows with m.

[8]Contrary to Figure 5.18, here we use a linear x-axis to emphasize the linear relation between the delay and c. This makes the behavior of small chunks difficult to observe on Figure 5.19, but the proportional relationship was also verified for small values.

In fact, the chunk is the unit of data exchange and a peer can re-transmit a chunk only if it has fully received it. To increase the chunk size increases the time needed to exchange a chunk, and as a consequence the diffusion delay increases. To increase the number of parallel upload connections, leads to a similar effect: the time needed to exchange a chunk increases with the number of connections because the upload bandwidth is shared with more nodes, and as a consequence the diffusion delay increases.

This result is consistent with theoretical results obtained in [90] where RTT is neglected and the chunk transmission time is simply considered inversely proportional to the sender's bandwidth. Under that framework, the minimal diffusion delay is given by:

$$d_{\min} = \frac{mc\ln(n)}{\ln(1+m)SR}. \tag{5.12}$$

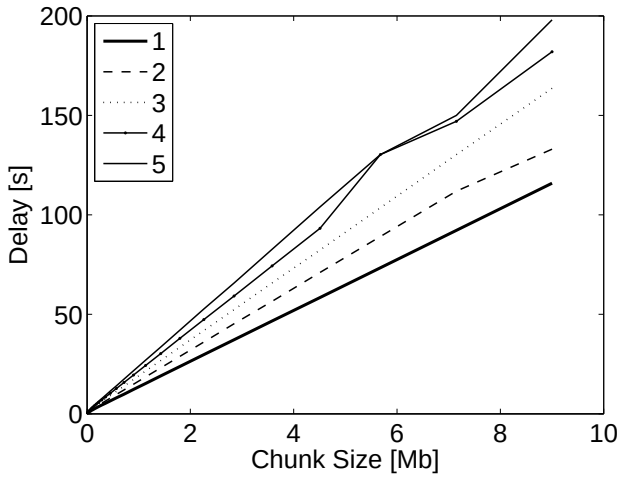


Figure 5.19 : Average diffusion delay as a function of the chunk size.

Overhead

The performance with respect to overhead, i.e. the difference between the throughput and goodput, is shown in Figure 5.20 (only the curves for $m = 1$ and $m = 5$ are displayed for legibility). For very small chunks, we have a non-intuitive trend, where as c grows, the goodput increases and the throughput decreases (or equivalently, the overhead decreases faster than the goodput increases). This process slows down so that at some point the throughput increases again. For big enough chunks, the overhead becomes roughly constant (for a given m), while the goodput becomes equal to the stream rate (meaning no missing chunks).

For very small chunks, chunk miss ratio is high, which, as mentioned earlier, come from the fact that not enough control messages can be sent. Asymptotically, we may imagine that only one control message per sent chunk is produced, resulting in an overhead/goodput ratio of $\frac{c_c}{c}$, where c_c is the size of a control message.

On the other hand, in the limit as the chunk size is increased, we may expect that a peer can send a number of messages per sent chunk that is proportional to the chunk characteristic time $\frac{c}{SR}$. This would result in an overhead ratio proportional to $\frac{c_c}{SR}$, and thus independent of c (but not of other parameters like the median RTT or m).

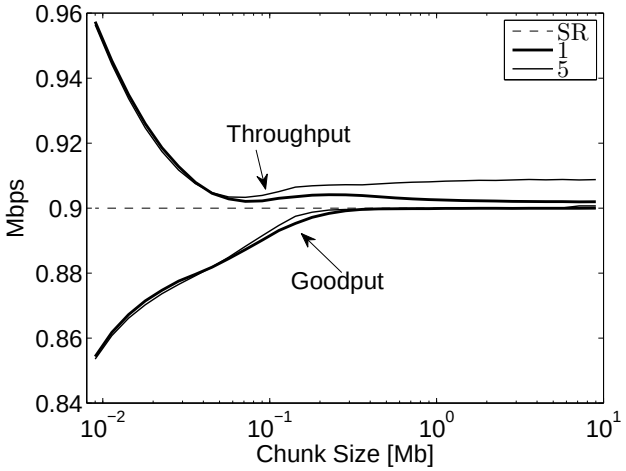


Figure 5.20 : Goodput and throughput as a function of the chunk size, the overhead being the difference. The stream rate SR is also indicated.

Suitable range for c

In light of the study above, there is a good order of magnitude for suitable chunk size. For the parameters considered here, c should be greater than 0.06 Mb (which corresponds to about 15 chunks per second) and smaller than 0.3 Mb (3 chunks per second):

- to send the stream at more than 15 chunks per second is good for the delay (which stays roughly proportional to c), but results in both an increase in throughput and a decrease in goodput;
- goodput and throughput are stationary for c greater than 0.3 Mb: using bigger chunks only means longer delay;
- between these values, the choice of c results in a chunk miss ratio/delay trade-off: smaller delay with some missing chunks or greater delay with no missing chunks. Choosing a precise value for c depends then on factors that will not be discussed here, such as the codec used, the required QoS, etc.

In our experiments the suitable range for chunk size begins when the chunk characteristic time ($\frac{c}{SR}$) has the same order of magnitude than the median RTT, and ends an order of magnitude later. We scaled the RTT distribution used in order to observe the evolution of the range with the median RTT. The results, reported in Figure 5.21, show that the range values are indeed roughly proportional to the median RTT.

Note that the lower bound of the suitable range gives an indication on the minimal delay that can be achieved without too much missing chunks and overhead. In section 5.4.3, we will see that enhanced diffusion techniques can help lower that bound.

We have performed simulations using various diffusion schemes, RTT and bandwidth distributions, number of parallel upload connections m, stream rate SR and so on. All results confirm the existence of a *suitable* range for c.

As an example, in the following we compare the suitable range of chunk size for two RTT values and three dissemination schemes. In particular, we consider the *rp/lu* scheme with *rp/lb*

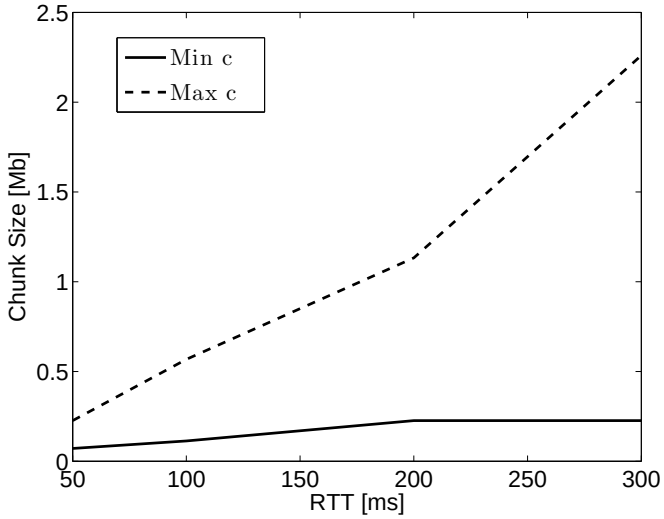
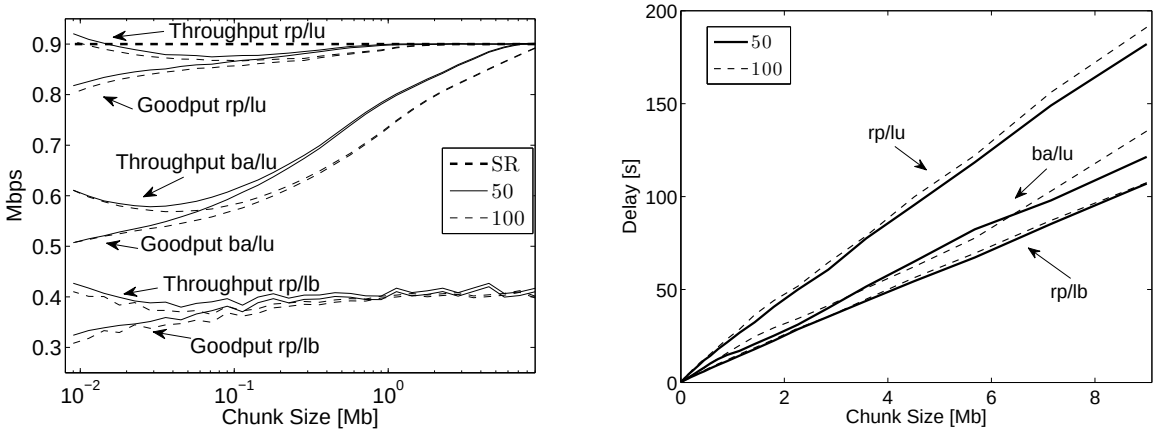


Figure 5.21 : Suitable range (for $m' = m$)

and *ba/lu*. Since the scenarios with homogeneous bandwidths are identical under the *rp/lu* and *ba/lu* schemes, we use an heterogeneous bandwidth distribution derived from [101]. We set $m = m' = 1$, and we plot the throughput, goodput and average delay for these cases using two values of average latency, RTT $= 50, 100$ ms (Figure 5.22).

Note that the scheme *rp/lb* suffers high chunk miss ratios for all values of chunk sizes considered. Indeed it has been shown in section 5.2 that this scheme performs poorly with respect to rate, while being optimal with respect to delay. The scheme *rp/lu* has fewer missing chunks, but higher delay, while the performance of *ba/lu* lies between the other schemes for both chunk miss ratio and delay.

However, beyond the fact that the chunk miss ratio/delay/overhead trade-off is closely related to the scheme, the striking observation is that all these schemes admit a similar suitable range for c, which seems to scale with the median RTT of the network. This supports our claim that the suitable range for c depends mainly on the median RTT and SR, the actual scheme being secondary.



(a) Average goodput and throughput

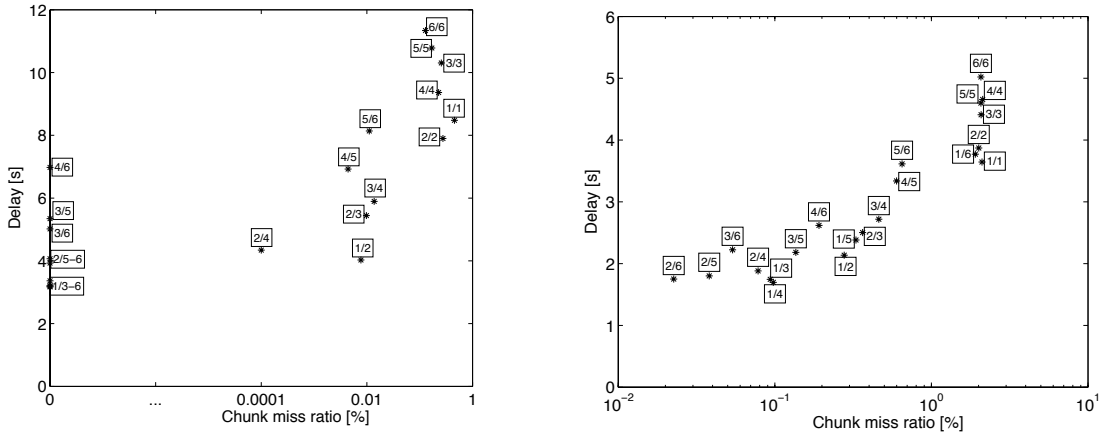
(b) Average diffusion delay

Figure 5.22 : *rp/lb*, *rp/lu* and *ba/lu* comparison

5.4.3 Size of Probe set

In the results presented so far, we have assumed that the number of simultaneous upload connections, m, is identical to the size of the probe set m'. We now consider the impact of probing more peers than the number of simultaneous chunks sent. A larger probe set affords a sender peer a higher chance to find a recipient peer for whom it has useful chunks (power of choices principle). However, it also increases overhead, and possibly delay.

Figure 5.23(a) plots the chunk miss ratio/delay trade-off for various m/m' pairs. The scheme is *rp/lu*, the bandwidth is homogeneous and the chunk size is set to $c = 0.15$ Mb (middle of the suitable range). The figure shows that using $m' = m$ is not optimal, and having a larger probe set, $m' > m$ significantly reduces both delay and missing chunks. The delay decreases from about 10 s for the $m = m'$ case, to less than 4 s for the $1/3, \ldots, 6$ cases (meaning $m = 1$ and $m' = 3, \ldots, 6$). With regards to the chunk miss ratio, there are some (m/m') pairs for which no missing chunks could be observed in our experiment: $1/3 - 6$, $2/5 - 6$, $3/5 - 6$, $4/6$. This suggests that a consequence of using $m < m'$ is a shift of the suitable range for c.



(a) $c = 0.15$ Mb (middle of the suitable range for $m' = m$)

(b) $c = 0.035$ Mb (below the suitable range for $m' = m$)

Figure 5.23 : m/m' chunk miss ratio/delay trade-off for two values of c.

In order to verify this interpretation, we now set $c = 0.035$ Mb, which is clearly below the suitable range observed in Figure 5.20 for $m = m'$. The results are shown in Figure 5.23(b).

We observe that no pair (m'/m) can achieve diffusion without missing chunks for such a small c, however the trade-offs are still worthwhile with respect to the $m = m'$ case: using $m/m' = 2/6$, we get a delay of 1.7 s with a chunk miss ratio of about 0.02 %, which represents an excellent trade-off, far better than the one observed for $c = 0.15$Mb and $m = m'$. This indicates that $c = 0.035$ Mb is definitively within the suitable range for $m/m' = 2/6$.

Also note how the relative efficiency of the various m/m' values is impacted by the choice of c: for instance, $1/6$, which is optimal for $c = 0.15$ Mb, performs rather poorly for $c = 0.035$ Mb. Although the results presented here refer to the *rp/lu* scheme, we performed experiments with other schemes and we observed similar trends, confirming that using a proper $m < m'$ can significantly improve the delay.

On the other hand, there is a price for going below the suitable range: for a given scheme, the overhead still depends on m' and c. For *rp/lu*, it stays close to the overhead displayed in

Figure 5.20 even for $m < m'$. So using small c with $m < m'$ can reduce the delay, but it requires more throughput.

5.5 Conclusion

In this chapter, we have considered the diffusion process in mesh-based peer-to-peer live streaming systems from a theoretical perspective. First, we have proposed an overview of schemes whose performance have been proven optimal in term of diffusion delay, rate or both. We have then identified a large set of practically interesting chunk distribution schemes, and provided explicit formulas to describe the diffusion functions of some of them. We have highlighted the existence of a rate/delay trade-off, and the significant impact a limited neighborhood and peer heterogeneity may have on the performance of the system.

We have shown that, in heterogeneous systems, the resources of the peers receiving the first copies of a given chunk strongly influence the final diffusion performance. In such scenarios chunk distribution schemes should therefore take into account the resources shared by nodes when performing the peer selection. Nevertheless, a certain level of agnostic selection is needed for the functioning of the system: a kind of equilibrium between aware and agnostic selection should be found that ensures a good utilization of the powerful nodes, while guaranteeing that weaker nodes are not excluded from the diffusion process.

We have proposed a model that explicitly takes this trade-off into account: such model is highly versatile and can encompass several existing resource aware algorithms. We derive explicit formulas for the diffusion function of a generic *resource aware peer / latest blind chunk* selection scheme, and highlighted the critical role the source peer selection policy plays on the performance of heterogeneous systems.

We have also analyzed the importance of crucial parameters, like the chunk size and the probe set size. We have shown the existence of a suitable range of chunk size that is mostly related to RTTs between nodes, and that a probe set larger than the maximum number of parallel upload connections may improve the performance a given scheme can achieve.

Chapter 6

Conclusion of PART I

Peer-to-peer live streaming has been an hot research topic since 2000, when first proposals to distribute a live event by means of P2P overlays appear. However, it has become a popular approach only recently, with the growing popularity of commercial applications like PPLive, UUSee, TVAnts, SopCast and so on. It turns out that the most popular systems, able to support more than 100000 simultaneous viewers for the same channel, are based on a mesh approach where the stream is no more forwarded as a continuous flow of data but is divided in a sequence of pieces. In such systems resource allocation algorithms drive the exchange of these pieces among peers in order to allow them to restrieve the continuous sequence and play out the stream. This content dissemination is managed by chunk exchange schemes that are executed locally at every node. These can be described by their chunk/peer selection policies.

Despite the popularity of such commercial systems, the main performance trade-offs of resource allocation algorithms in mesh-based live streaming applications are not completely understood yet. Diffusion schemes are often not known in details, so that it is not possible to fully analyze their behavior from measurement analysis. Moreover, an analytical model is difficult to derive because of the complexity of the whole application.

In this thesis part, we have precisely considered this problem and focused on the performance evaluation of chunk distribution algorithms for mesh-based live streaming applications. Thanks to the experimental analysis of PULSE, a mesh-based live streaming system we designed and developed, we have shown the effectiveness of a mesh-incentive approach for the deployment of a peer-to-peer live streaming application. We have highlighted that nodes contributing with more resources to the system are advantaged by achieving lower reception delays. The whole system benefits as well, because the placement of more resourceful nodes at the top of diffusion trees decreases their length, leading to lower reception delays for all users.

We have shown that the distribution trees of our mesh-incentive system have some emerging characteristics that are similar to the theoretical properties of tree-based live streaming applications. As a consequence, PULSE achieves diffusion performance comparable to structured systems. However, such performance is still far from the theoretical optimal one.

We have then considered the building blocks of the diffusion process: the chunk/peer selection policies. We have derived some simple and practically interesting diffusion schemes, and we have analyzed them by means of theoretical analysis and simulations. We have shown a *diffusion rate/delay* trade-off arises, and that optimal rate dissemination within an optimal delay can

be achieved by some schemes. We have also highlighted the crucial role that the overlay and the upload bandwidth distribution of nodes play on the final diffusion performance.

In heterogeneous systems, we have highlighted that the quality of a given chunk's diffusion is mostly affected by its early dissemination. We have derived a model able to represent several resource aware diffusion algorithms and highlighted a *trade-off between aware/agnostic peer selection*. Equilibrium between these two forces is needed in order to give priority to more resourceful nodes while not excluding the other peers from the diffusion process. Moreover, the capacity and selection policy of the source can have a tremendous impact on the diffusion performance a given system can achieve.

Finally, we have shown that, despite the theoretical optimal performance of some diffusion schemes, a wrong choice of system parameters, like the chunk size and the probe set size, may lead to sub-optimal diffusion performance. In particular, we have shown the existence of a suitable range of chunk size where optimal performance can be achieved as a function of pairwise delays between nodes, and we have highlighted how the use of probe sets larger than the maximal number of parallel upload connections can extend this suitable range.

Some problems are still open for mesh-based live streaming systems and are subject of current or future investigation:

- *Performance of optimal diffusion schemes in real systems*. Optimality results and performance trade-offs of the so called *epidemic-style* diffusion algorithms have been studied by means of theoretical and simulations. An interesting question is: how do such schemes behave in real implementations? What modifications do they need to overcome practical constraints? First implementations ([90, 129]) have shown the effectiveness of some optimal or near-optimal algorithms, but others should be tested as well, and a complete comparison under different network conditions, peer dynamics and network topologies is still missing.

- *Optimal diffusion delay in heterogeneous environment*. Schemes that can achieve optimal diffusion rate within an optimal delay have been derived for homogeneous systems, where all peers have the same upload capacity. As concern heterogeneous systems, rate-optimality has been proven for the so called *most deprived peer / random useful chunk* [72]. First results regarding delay optimality in such heterogeneous networks are presented in [65, 75]. However, the exact delay bounds that can be achieved by mesh-based schemes in heterogeneous systems are still not clear, and, as a consequence, a delay optimal scheme for heterogeneous network is still missing.

- *Multiple streams*. The results presented so far have been derived by considering only one stream. New results and trade-offs may arise by introducing the multiple streams constraint to the problem in mesh-based live streaming systems.

- *Network locality and awareness support*. A recent measurement analysis highlights the lack of strong locality and network awareness mechanisms in commercial live streaming systems [28]. A simple technique to add some latency awareness has been proposed in Chapter 4. A centralized ISP-managed approach, called P4P, has firstly been proposed in [122], while an ISP-friendly scheduling mechanism has been presented in [91]. Network awareness and locality mechanisms are currently subject of research of projects, like Napa-Wine [79], or standardization efforts, like ALTO [120].

- *Overlay/underlay interaction*. The impact of mesh-based applications on the underlying network resources is not completely understood, and so is the interaction between resource management at network layer and application layer. Another interesting aspect is the impact of multiple overlays, with common or different resource allocation techniques, in presence of a congested underlying network.

- *Stream coding*. In this chapter, we assumed simple source coding techniques, like FEC, in order to decode the stream even if some chunks are missing. However, layered video coding [66], or network coding [118, 119], have been shown efficient for P2P live streaming. It could be interesting to extend this analysis to other coding techniques and to derive exchange algorithms taking into account the coding used when performing the selection.

Part II

Video-on-Demand Streaming

Chapter 7

Introduction

In the previous part of the thesis we have considered the diffusion of live streaming through peer-to-peer overlays. In this part we consider another streaming application: the *on-demand streaming*. Even if we are in a similar context, the algorithms and techniques developed and analyzed for the live distribution cannot be directed applied to the on-demand problem.

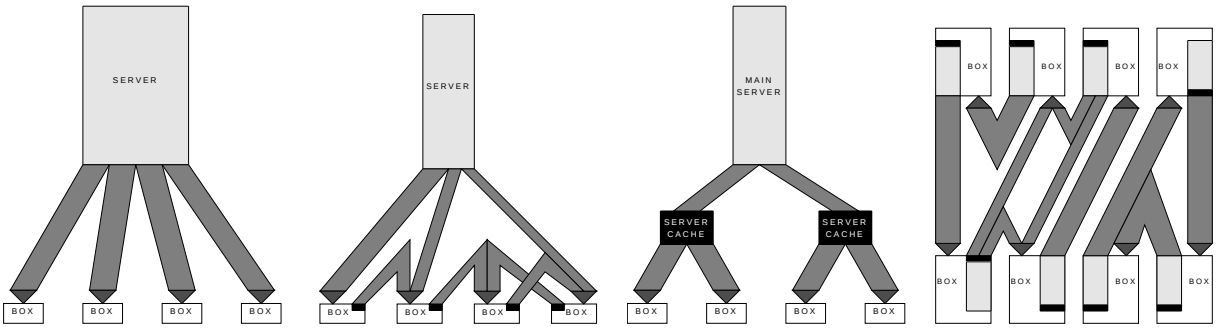
In on-demand streaming:

- the multimedia contents are completely available before the actual use of the service. They can therefore be stored, introducing the storage capacity constraint to the problem.
- users are not synchronized, so those stored multimedia contents should always be available to all users. Moreover, this lack of synchronization adds complexity to the exchange of contents between the different clients of the service.

The two most popular on-demand streaming services are *video-on-demand (VoD)* and *user generated content distribution (UGC)*. In a *VoD* service there is usually a catalog of media files (i.e. videos) that is offered to users: this catalog should be as larger as possible in order to attract all kinds of customers, and videos should be always available, so that a subscriber can access any video at anytime. This service typically concerns movies or TV series, and it is offered by several providers like PPLive [95], and Orange [103]. A *UGC* service offers customers the possibility to upload their own home videos or audio files that become accessible by any other user. Such service is usually offered by large-scale Internet application providers; popular examples of UGC applications are YouTube [124] and DailyMotion [33]. UGC differs from VoD mainly in the rate at which multimedia contents are updated, and in the content popularity distribution. In fact, contents of a UGC service are updated much faster than in a VoD system, and their popularity distribution is more skewed [19]. This requires more complex techniques for the storage and maintenance of the content catalog in a UGC service.

Several architectures and resource allocation algorithms have been designed to deal with storage and bandwidth constraints of on-demand streaming.

The *client-server* architecture (Figure 7.1(a)) is the simplest approach: this is suitable for small scale applications with a limited number of users and a small catalog of media files. In fact, the storage and bandwidth constraints limit the number of users that can access the service and



(a) Client-Server (b) Peer-assisted (c) CDN (d) Peer-to-Peer

Figure 7.1 : Architectures for on-demand streaming. We suppose users access the service by means of a device called box (e.g. set-top box).

the number of contents that can be stored. Moreover, the existence of a single point of failure (i.e. the server) implies the use of backup mechanisms to guarantee video availability, and the amount of storage and bandwidth capacities required are so large that it becomes expensive for service providers. Under the client-server architecture, resource allocation algorithms are very simple because there is a central entity in charge of storage and distribution of the contents.

In the *peer-assisted* architecture (Figure 7.1(b)) clients actively participate in the content diffusion process. They devote part of their storage capacity to the content they are playing[1], and share part of their network bandwidth to upload this content to other users. *Connection management* algorithms are in charge of the management of the network capacity in order to connect a user demanding a given content to the entities (other users and server) storing this content. Some *content allocation algorithms* are also needed to manage the storage capacity shared by users. Prefetching mechanisms may be applied to proactively push part of the contents to users even if they are not requesting them.

We have shown in Chapter 2 that the peer-assisted approach provides scalability in term of number of customers, while reducing the costs for the service provider. In fact, the bandwidth needed to set up the application is in large part provided by the users themselves. On the other hand, the catalog size scalability is not straightforward because there should be a central entity storing the whole video catalog.

This approach has been widely considered in literature. The PPLive VoD service is based on a peer-assisted architecture and have been analyzed in [52]; several statistics on the user behavior and various performance metrics are presented, as well as a discussion on the observations derived from the system design. Other works target to reduce the load at central server. In [51] an analysis of MSN Video traces shows that the peer-assisted approach dramatically reduces this load, particularly if prefetching techniques are employed. BitTorrent-like mechanisms are proposed and evaluated in [23, 87], while [6] considers how cable companies can leverage this load with simple techniques and current hardware. In [7], scheduling techniques and network coding are analyzed for the diffusion of a video using a mesh overlay.

The *Content Distribution Network (CDN)* architecture (Figure 7.1(c)) consists of a set of servers (or server farms) spread worldwide, storing all or part of the video catalog. Users then connect

[1]We call *cache* the storage capacity devoted to store the video downloaded by a user; the number of cached contents depends on its size. In figure the cache is represented in black color.

to one of the servers to retrieve the desired content. The choice of the server may depend on several factors: it may be the less loaded one, or the closest to the user, and so on. Under such an approach, *connection management algorithms* are needed for the connection of a user to a given server, and *content allocation techniques* are required to manage content replicas at the different servers. This kind of approach has been widely deployed by several providers, like Akamai [109]. In the on-demand streaming context, the Akamai CDN is for example used for the storage and distribution of YouTube contents. With a CDN the provider costs are reduced with respect to the client-server architecture: bandwidth and storage requirements at every mirror server are lower than in a central server case, and it is cheaper to buy several smaller amount of resources than to concentrate everything in a single point.

A *fully peer-to-peer* architecture (Figure 7.1(d)) has been first proposed for on-demand streaming by Suh *et al.* [107] under the name *push-to-peer*. There is no more a central entity storing the content catalog, but media files are stored in a decentralized fashion at users. The contents are proactively pushed to customers, during the night or outside peak hours of the day. These customers share part of their storage capacity: like in the case of storage at mirror servers in a CDN architecture, *content allocation* algorithms are responsible for the distributed storage at users under this approach. A given user then retrieves the desired content from other users storing it: this is handle by means of *connection management* techniques similarly to the peer-assisted architecture. Apart from the capacity devoted to the catalog storage[2], users can share part of their memory to store the content they are playing (or the latest downloaded contents), so that they can provide it to other customers.

We have seen in Chapter 2 a peer-to-peer approach can provide scalability in term of network bandwidth; however, in such storage constrained scenario, the peer-to-peer approach can also provide scalability in term of memory space to store the media catalog: the storage capacity increases with the number of users. Moreover, such storage capacity is not expensive for the provider because it consists in only some tens of GB at every user, which are quite cheap nowadays. These storage and bandwidth capacity requirements are already available in the set-top boxes installed at user home, making these devices the natural choice for the deployment of a peer-to-peer on-demand service.

To the best of our knowledge only few works consider the fully peer-to-peer architecture for on-demand streaming. Suh *et al.* [107] propose and model resource allocation and connection management techniques by focusing on network bandwidth constraints, while [54] proposes an architecture and some scheduling policies.

Contributions

We have mentioned that the allocation of network and storage resources in an on-demand streaming system is driven by *connection management and content allocation algorithms*. In this part of the thesis, we consider resource allocation in a peer-to-peer architecture. In Chapter 8 we derive from [107] a model that considers both network bandwidth and storage capacity constraints. First, we derive bounds on the achievable catalog size in systems where the total upload capacity is scarce, i.e. equal to the amount required to serve the video requests. Then, we show that catalog size scalability can be achieved as soon as peers have an average upload

[2]In figure this memory space is represented in light gray color.

capacity slightly larger than the content rate. In Chapter 9 we tackle the problem from a more practical perspective and we propose some simple content allocation and connection managements techniques. We analyze the performance of such algorithms by means of simulations and experimental evaluation. These techniques can serve as guidelines for the design and parameter tuning of peer-to-peer on-demand streaming systems. These policies may also be useful to other architectures, like peer-assisted or CDN.

Chapter 8

Catalog size in distributed Video-on-Demand systems

In this chapter we analyze the number of videos (catalog size) a Video-on-Demand system can provide as a function of the bandwidth and storage constraints. In particular, we consider a fully peer-to-peer architecture where customers of the service are also in charge of storing the whole video catalog, and actively participate in serving video requests generated by other users. Nevertheless, our results may be applied to a UGC service, and to other architectures where scalability is limited by these constraints.

As far as we know, Suh *et al.* [107] are the only ones to study resource allocation algorithms for peer-to-peer architectures by means of a theoretical model. They focus on coding and bandwidth constraints but do not consider the storage limitations. We extend their model to take also the storage capacity into account, and we derive bounds on the number of contents a VoD system can store and serve to users.

Contents of this chapter are a joint work with Yacine Boufkhad, Fabien Mathieu, Fabien de Montgolfier and Laurent Viennot and are partially presented in [142, 136].

8.1 Scarce upload capacity

We consider a system composed of n entities called boxes, with both storage and network capabilities. For instance, this is representative of set-top boxes installed at user home. Some videos are pre-fetched in the boxes' storage space, and every time a user wants to play a video, his box downloads it from other boxes on the fly.

The service provides a catalog of m distinct videos. For simplicity, we assume that all videos have the same duration T, the same stream bit rate SR (for simplicity normalized to $SR = 1$), and therefore require the same amount of memory space to be stored.

We suppose a homogeneous system where every box has a storage capacity of d videos and an upload capacity equivalent to u video streams (for instance if $u = 1$ a box can upload exactly one stream). Additionally, we suppose that each box is always available and we assume that the physical download capacity is not a bottleneck. Note that the catalog size under such hypothesis is bounded by $m = nd$.

n	Number of boxes in the system
m	Number of videos stored in the system (catalog size)
SR	Video stream rate
d	Storage capacity of a box (in # of videos)
u	Upload capacity of a box normalized w.r.t SR
s	Number of parallel upload connections per box
c	Number of stripes per video (a video can be viewed by downloading its c stripes simultaneously)
T	Video length
r	Number of simultaneous video requests

Table 8.1 : Parameters for scarce upload system analysis.

These boxes are used to serve a sequence of video requests. As new requests arrive old requests terminate, and the number of simultaneous requests at a given time is supposed to remain bounded by r. For each request, a *connection management* algorithm designates the set of boxes that will collectively upload the requested video. This algorithm can be *static* when the designated boxes remain the same until the end of the request, or *dynamic* when the designated boxes remain the same until arrival of the next request.

To enable collective upload of a video, each video may be divided in c *stripes* of equal size using some balanced encoding scheme. The video can then be viewed by downloading simultaneously the c stripes at rate $1/c$ (playback rate is normalized to 1). A very simple way of achieving striping consists in splitting the video file in a sequence of small packets. Stripe j is then composed of the packets with sequence number i so that $j = i \ modulo \ c$. We do not discuss further how striping can be achieved but advanced striping techniques may for example include some redundancy or coding techniques in order to make the playback feasible even if only $c' < c$ stripes are available (like [15, 78]).

We suppose that a box may open at most s simultaneous connections for uploading video data. The main reason is that we assume that a downloader may only open a limited number of connections to ensure a low startup time and manageable protocols. A balanced scheme must then result in a bounded number of upload connections per box. Another reason is that the total goodput decreases if too many connections are involved [9]. We additionally assume that the connections of a box fairly share its bandwidth as it would typically be the case for simultaneous TCP connections. Parameters are summarized in Table 8.1.

Main results

We focus on systems where the total upload capacity is equal to the amount needed to satisfy the video requests, i.e. $un = r$. First, we consider in Section 8.1.1 an optimal allocation algorithm, the *full striping* [107], that provides the maximum catalog size $m = nd$. However, we show this bound cannot be achieved if the number of connections per box is limited. In section 8.1.2 we propose a cyclic allocation scheme allowing to store a catalog of size $m = ds\frac{n}{r}$ in case of s parallel upload connections per box, which can be increased to $m = (d-u)s\frac{n}{r} + un$ if requests are for distinct videos. We then show in section 8.1.3 that $m = (d-u)s\frac{n}{r} + un$ is indeed the excat upper bound on the number of videos the system can store, if $un = r$ and boxes have a limited number of upload connections.

8.1.1 Full striping

As stated in [107] by Suh *et al.*, there exists a simple optimal scheme for video allocation when the number of simultaneous connections is unbounded: the *full striping*. Assume each video can be split in n stripes. A system with n boxes can then store $m = dn$ videos by allocating to each box one stripe of rate $1/n$ for each video. Any request sequence for r videos will then result in a demand of r stripes per box. They can always be served as long as $u \geq r/n$. Notice that dn is a trivial upper bound on the number of videos that can be stored in the system. However, we show that a system with scarce upload capacity and limited number of simultaneous connections cannot offer as many videos as it can store.

8.1.2 Cyclic allocation

We propose now an allocation scheme that provides a lower bound on the catalog size m for a static download scheme, in case of limited number of upload connections per box s. We show this lower bound increases in the special case of distinct video requests and dynamic download scheme.

Theorem 8.1. *[142] If $un = r$ and $s \leq r$, it possible to offer $ds\frac{n}{r}$ videos. Moreover, it is possible to offer $(d-u)s\frac{n}{r} + un$ videos when requests are distinct.*

Consider a coding scheme with $c = sn/r$ stripes per video. For each $0 \leq i < c$, store stripe i on the n/c boxes with number j such that $j \equiv i$ modulo c. Requests are then served by downloading video number j from the boxes with number $j, j+1, \ldots, j+c-1$ (modulo n). As a consequence each box has to serve at most $rc/n = uc = s$ demands of upload $1/c$. This download scheme allows to store a catalog of $dc = dsn/r$ videos, and is static. Note that $c \leq n$ is a necessary condition to respect the upload constraints of boxes. It is possible to achieve a slightly better bound of $d(c+1)$ if the connection from a box to itself is not accounted in the simultaneous connection constraint.

When a request consists in r different videos, a catalog of size $(d-u)s\frac{n}{r} + un$ videos in the system can be achieved as follow. Store $(d-u)s\frac{n}{r}$ videos according to the previous scheme plus un videos "uniquely" stored in the following sense: stripe i of video j is stored on box number $i+j$ modulo n. A request for $r = un$ videos is satisfied by allocating first the demands for uniquely stored videos. Each remaining video v can then be served by the c boxes storing a uniquely stored video which is not part of the request. If ever this uniquely stored video is then requested, v has to be viewed from another set of boxes with free capacity. The download scheme is thus dynamic.

8.1.3 Upper bound

Theorem 8.2. *[142] In the case where $un = r$, the number m of videos offered by the system is at most $m = (d-u)s\frac{n}{r} + un$.*

The main idea of the proof is the following. Consider a set-top box b storing data from i different videos. The number of videos that are not stored in b is at most $u(n-1)$, otherwise a request

for $un = r$ videos not stored in b would fail because of a lack of upload capacity. It is possible to show that the maximum number of videos stored in b is $i \leq (d-u)s\frac{n}{r} + u$, otherwise box b will upload less than r/n and the system will not have the capacity to serve the r videos. This brings to a catalog of size $m \leq (d-u)s\frac{n}{r} + un$. The proof is detailed in [142].

Note that we get a similar bound in a system where downloaders are constrained by a maximum number s' of simultaneous connections. If all peers open at most s' download connections, then some box has $s'\frac{r}{n}$ upload connections at most. With the same arguments, we then get $m \leq (d-u)s' + un$.

8.2 Scalable catalog size

In this section we consider systems where the upload capacity is larger than the amount required to serve the r video requests. In particular, we show catalog of size linear in n can be provided as soon as the average bandwidth of a box is slightly larger the stream rate. Our results apply in the case where every box performs a video requests, i.e. $r = n$.

For this analysis we have to extend the model used in the previous section. We now suppose very box b has a physical storage capacity of $\mathcal{D}_b$, which corresponds to $d_b := \frac{\mathcal{D}_b}{T \cdot SR}$ videos. In this case, d represents the average storage capacity of a box. In addition to d_b, we assume that a box b has a cache where it stores all data already downloaded of the video it is currently viewing. If a box plays videos one after another, the cache then contains the end of the previous video and the beginning of the current one according to a *Least Recently Used* cache removal policy.

As for the storage constraint, we suppose a box b has a physical upload capacity $\mathcal{U}_b$. We define the relative capacity as $u_b := \frac{\mathcal{U}_b}{SR}$, and in this case u indicates the average upload capacity of a box. Again, we assume that the physical download capacity is not a bottleneck.

We suppose that a box can upload a stripe only if it can allocate an upload capacity of SR/c (at least) to the stripe, i.e. only if it has enough capacity to upload the stripe at its stream rate. This means that a box can upload at maximum $s_b = \lfloor u_b c \rfloor$ stripes in parallel.

For convenience, we use a discrete round-based model, where the time unit is the time necessary for a box to establish a connection and start data transfer. New requests may arrive within any round, and boxes may establish new connections in the following rounds. The start-up delay is the maximum number of rounds elapsed between arrival and the beginning of the playback of the video. A constant number of rounds only is allowed so that start-up delay remains constant. Furthermore, we suppose that the number of requests for a given video increases at most exponentially with time: if $f(t)$ denotes the size of a *swarm*, i.e. the population of boxes viewing the same video, then we assume $f(t+i) \leq \lceil \max\{f(t), 1\}\, \mu^i \rceil$ for some $\mu > 1$. We call μ the maximal *swarm growth*: the size of a swarm increases by a factor at most μ at each time round.

To summarize, we call (n, u, d)-*video system* a set of n collaborative boxes with average upload capacity u and average storage capacity d. Such a system is *homogeneous* if all boxes have same upload capacity and same storage capacity, i.e. for all b, $u_b = u$ and $d_b = d$. It is *proportionally heterogeneous* if $\frac{u_b}{d_b} = \frac{u}{d}$ for every box b. We say that an (n, u, d)-video system *achieves* catalog size m if it is possible to store m distinct videos on the boxes so that any sequence of requests of at most one video per box can be satisfied as long as the maximal swarm growth μ is respected.

n	Number of boxes in the system
m	Number of videos stored in the system (catalog size)
SR	Video stream rate
d_b	Storage capacity of b (in # of videos)
d	Average storage capacity of a box (in # of videos)
$\mathcal{D}_b$	Storage capacity box b (in bytes)
$\mathcal{D}$	Average storage capacity of a box (in bytes)
k_v	Number of copies of video v
k	Average number of copy per video ($km \leq dn$)
u_b	Upload capacity of box b normalized w.r.t SR
u	Average upload capacity of a box normalized w.r.t SR
$\mathcal{U}_b$	Upload capacity of box b in bytes per second
$\mathcal{U}$	Average upload capacity of a box in bytes per second
s_b	Number of stripes box b can upload in parallel
s	Average number of stripes a box can upload in parallel
c	Number of stripes per video (a video can be viewed by downloading its c stripes simultaneously)
T	Video length
T_S	Start up time
μ	Swarm growth bound: if a swarm has size p at round t, its size is less than μp at round $t+1$
ℓ	Minimum amount of data of a given video stored in a box
r	Number of simultaneous video requests

Table 8.2 : Parameters for the analysis of catalog size scalability.

An *allocation* is the process of storing stripe replicas into boxes statically. We define ℓ as the minimum amount of data of a given video stored in a box. When splitting videos into c stripes, we get $\ell = \frac{1}{c}$. The scalability condition for c translates into $\ell = \Omega(1)$, i.e. a video cannot be split into infinitely small pieces as n increases.

All parameters of this extended model are summarized in Table 8.2.

Main results

First note that $u \geq 1$ is a natural requirement for catalog scalability if all boxes may play videos at the same time. Suppose $u < 1$ and consider a sequence of requests where each box always plays a video it does not possess. As observed in Chapter 2 the aggregated download rate should be equal to n, whereas the aggregated upload rate is $un < n$ which is not sufficient. As minimum amount of data per video stored at a given box is ℓ, each box b stores data of at most $\frac{d_b}{\ell}$ videos. Set $d_{\max} = \max_b\{d_b\}$. If $m > \frac{d_{\max}}{\ell}$, then for each box b, there always exists a video v not possessed by b, i.e. b stores no data at all from v. Therefore, in order to satisfy the video requests of all boxes we must have $m \leq \frac{d_{\max}}{\ell}$. In this case, catalog size is thus constant as long as $d_{\max} = O(1)$ and $\ell = \Omega(1)$.

In contrast, our main result states that it is indeed possible to have a linear catalog size as soon as $u > 1$. We propose in Section 8.2.1 two random video allocation techniques where each video is split into c stripes of rate $\frac{1}{c}$, which are replicated a constant number of times k. In Section 8.2.2 we consider $k = O(\log_u d')$, where $d' = \max\{d, u, \exp(1)\}$. Theorem 8.4 formally states

that catalog size $\Omega\left(\frac{(u-1)^2 \log \frac{u+1}{2}}{u^3} \frac{1}{\mu^2} \frac{dn}{\log d'}\right)$ (linear in n since μ, d' and u are constant) can be achieved under these conditions if a random permutation allocation is used. Finally, the result is generalized to the case of a heterogeneous system in Section 8.2.3. A solution is proposed to overcome the difficulty caused by the boxes having upload less than 1. It consists in relaying the demands of these boxes through the boxes having enough upload bandwidth.

Our approach consists in applying together maximum flow arguments and the probabilistic method to show that a valid allocation of videos can be found with high probability. For that purpose we have to show that all the graphs of "who gives what" encountered in the infinite sequence of requests have some expander property. This is possible through the combination of algorithmic arguments concerning restrictions on how requests are made and probabilistic arguments on how videos are allocated. This result does not yield directly a practical distributed algorithm. However, it shows that scalable video on demand is theoretically feasible for $u > 1$.

8.2.1 Preliminaries

We now present the basic requirements for achieving a given sequence of requests by considering the graph linking each request to the boxes that possess the corresponding data. We first briefly present how videos can be randomly placed in the system when using c stripes of rate $\frac{1}{c}$ per video and k[1] replicas per stripe.

Random allocation

Random allocation consists in storing $k \geq 1$ replicas of each stripe into k boxes chosen randomly, either independently or according to a random permutation. For the sake of simplicity, we assume that $k = dn/m$ is an integer. A *random independent allocation* consists in selecting independently for each stripe replica a box with probability proportional to its storage capacity. (The process is stopped as soon as a replica falls in a completely filled-up box). Alternatively, a *random permutation allocation* consists in copying each stripe into k boxes such that each box contains exactly dc stripe replicas. We model this through a random permutation π of the $kmc = dnc$ stripe replicas into the dnc storage slots of the n boxes together: replica i is stored in slot $\pi(i)$ (the d_1c first slots fall into the first box, the d_2c next slots into the second box, and so on). The highest catalog size is obtained for the smallest possible value of k.

We call *random allocation* the process consisting in encoding each video into c stripes and storing k replicas of each stripes randomly on boxes, either according to a random permutation, or a random independent allocation. The latter is simpler but may lead to unbalanced storage load at boxes. In order not to exceed the capacity of any box with high probability, we need an additional requirement on the number of stripes, which increases the number of copies per video needed with respect to a random permutation allocation. On the contrary, this last fully utilizes the storage capacity of all boxes leading to a larger catalog of videos.

[1] We assume that all videos are replicated the same number of times i.e. $\forall v, k_v = k$.

Connection matching

We model the problem of finding connections for downloading the video stripes at a given time as a maximum flow problem. Let W denote the set of boxes and consider the set Y of requested stripes at time t. We define G as the complete bipartite graph from Y to W. A *connection matching* is a matching of requests against boxes possessing the necessary video data (from the original allocation or in their caches), so that each box b has degree at most $\lfloor u_b c \rfloor$ and each stripe request has degree 1. Wiring connections according to such a matching allows to satisfy requests at round $t+1$ as each stripe has rate $\frac{1}{c}$. Finding such a connection matching is modeled as a maximal flow computation in the bipartite graph G.

Maximum flow feasibility

We can characterize the existence of a connection matching as follows. Let $B(x)$ denote the neighbors of a request $x \in Y$ in G, i.e. the set of boxes possessing data for x at time t. More generally, for a subset $X \subseteq Y$ of requests, let $B(X) = \cup_{x \in X} B(x)$ denote the set of boxes possessing data for any request $x \in X$. For a set $E \subseteq W$ of boxes, let $U_E = \sum_{b \in E} u_b$ denote its overall capacity. We can then state the following lemma (which is a simple generalization of Hall's theorem).

Lemma 8.3 (Min-cut max-flow) *[136] A connection matching for satisfying requests at the next time round exists iff for all $X \subseteq Y$, $U_{B(X)} \geq \frac{|X|}{c}$ where $U_{B(X)} = \sum_{b \in B(X)} u_b$.*

We call *request obstruction* a subset X of requests such that $U_{B(X)} < \frac{|X|}{c}$. We are indeed interested in the multiset $M(X)$ of stripes requested in X. We extend this notion to any multiset of stripes: we call *obstruction* a multiset σ of stripes such that there exists a sequence of video demands that has been satisfied up to time t and where a subset X of requests at time t satisfies $M(X) = \sigma$ and $U_{B(X)} < \frac{|X|}{c}$. Clearly, Lemma 8.3 implies that any sequence of demands can always be satisfied iff there exists no obstruction.

We can then bound the probability that a given random allocation can be defeated as follows. We denote by N_k the random variable defined as the number of obstructions (among all possible subsets of at most nc stripes) in a permutation allocation chosen uniformly at random from the set A_k of all possible random allocations for a given k (and a given type of allocation: permutation or independent). Let $\mathcal{O}$ be the set of multisets of stripes with cardinality at most nc. For some allocation a and a multiset of stripes σ, we denote by $I(a, \sigma)$ the indicator variable that is equal to 1 if σ is an obstruction and 0 otherwise. Using the first moment method, we can bound $P(N_k > 0)$ (the probability that a random allocation admits at least one obstruction):

$$\begin{aligned} P(N_k > 0) &\leq E(N_k) \\ &= \frac{\sum_{a \in A_k} \sum_{\sigma \in \mathcal{O}} I(a, \sigma)}{|A_k|} \\ &= \sum_{\sigma \in \mathcal{O}} \frac{\sum_{a \in A_k} I(a, \sigma)}{|A_k|} = \sum_{\sigma \in \mathcal{O}} P(\sigma) \quad (1) \end{aligned}$$

where $P(\sigma)$ is the probability for some multiset of stripes σ to be an obstruction in a randomly chosen allocation. As we shall see, for sufficiently high values of u and k, the expectation of the

number of obstructions is bounded by $O\left(\frac{1}{n^\lambda}\right)$ for some positive λ and then with high probability the number of obstructions in a randomly chosen allocation is zero.

8.2.2 Homogeneous systems

We can now state the main theorem in the homogeneous case where all boxes have the same bandwidth and storage capacity.

Theorem 8.4. *[136] Given $u > 1$, consider a homogeneous (n, u, d)-video system. With high probability, a random permutation allocation with $c > \frac{2\mu^2-1}{u-1}$ and $k \geq 5\nu^{-1}\frac{\log d'}{\log u'}$ for $\nu = \frac{1}{c+2\mu^2-1} - \frac{1}{uc}$, $u' = \frac{1}{c}\lfloor uc \rfloor$ and $d' = \max\{d, u, \exp(1)\}$ allows to successfully satisfy any sequence of requests with maximal swarm growth μ. As a consequence, the system can achieve catalog size $\Omega\left(\frac{(u-1)^2 \log \frac{u+1}{2}}{u^3}\frac{1}{\mu^2}\frac{dn}{\log d'}\right)$.*

The result holds also for a random independent allocation with same bounds for c and k. However, in the random independent case, box storage loads may be unbalanced. To avoid to exceed the capacity of any box with high probability, an additionally condition on the number of stripes is required $c = \Omega(\log n)$. For large n, we have $u' \geq \frac{u}{2}$, $\nu^{-1} \sim \frac{uc}{u-1}$ and $k = O\left(\frac{u}{u-1}\frac{\log d'}{\log \frac{u}{2}}\log n\right)$ is then sufficient to obtain catalog size $\Omega\left(\frac{(u-1)\log\frac{u}{2}}{u}\frac{d}{\log d'}\frac{n}{\log n}\right)$.

The proof mainly relies on two arguments. First, a request strategy is proposed to cope with highly demanded videos. Second, a randomized argument bounds the probability that a request for various videos cannot be satisfied by the boxes storing them according to the random allocation scheme.

The request strategy is the following. Consider a box b where the user demands a video v at time t. A *preloading* request for one stripe v' of v is first issued at time t. Then $c-1$ *postponed* requests are made for the $c-1$ remaining stripes of v at time $t+1$. The start-up delay for playing a video is thus 3 time rounds. To balance preloading requests, we use a counter for each video v to give successive numbers to boxes entering the swarm of v. The pth box then preloads stripe number p modulo c so that all stripes of a video are equally preloaded. We will see that this strategy allows to manage a large swarm of growth μ as long as the number of stripes is sufficiently large. The proof relies on Equation 8.1 that consists in bounding $P(\sigma)$ for every multiset σ of size at most nc.

As a first step an upper bound of $P(\sigma)$ that depends only on the number of stripes in σ and the number of pairwise distinct stripes among them is derived in Lemma 8.5, 8.6, 8.7. To this purpose, it is necessary to estimate the number of boxes that can serve the requests made during a time interval $[t-T, t]$ thanks to the following lemma.

Lemma 8.5. *[136] At time t, consider any subset X of stripe requests made in $[t-T, t]$. Let $i = |X|$ denote the size of X and let i_1 denote the number of pairwise distinct stripes requested in X. Then the set $B(X)$ of boxes that can serve requests in X satisfies $|B(X)| \geq \frac{i-(c+2\mu^2-1)i_1}{c+2(\mu^2-1)}$.*

The following lemma bounds from above the probability that a set of pairwise distinct stripes are allocated to the same set of p boxes in a random permutation allocation. It is also trivially satisfied if stripe replicas are placed according to a random independent allocation rather than a random permutation.

Lemma 8.6. *[136] Consider a random permutation allocation of $kmc = dnc$ stripe replicas into the dnc memory slots of n boxes. The probability that ki given replicas fall into p given boxes with $dpc \geq ki$ is less than $\left(\frac{p}{n}\right)^{ki}$.*

It is now possible to bound the probability that a multiset of at most nc stripes is an obstruction.

Lemma 8.7. *[136] Let σ be a multiset of stripes of size $i \leq nc$. Let i_1 be the number of pairwise distinct stripes in σ. The probability $P(\sigma)$ of σ to be an obstruction is at most $P(\sigma) \leq \left(\frac{u'nce}{i}\right)^i \left(\frac{i}{u'cn}\right)^{ki_1}$. In addition, $P(\sigma) = 0$ when $i_1 \leq \nu i$.*

Note that the assumption on c implies $uc > c + 2\mu^2 - 1$. We thus have $0 < \nu < 1$ as $\nu = \frac{1}{c+2\mu^2-1} - \frac{1}{uc}$.
From Equation 8.1, and using Lemma 8.7, it is possible to show that the probability that an obstruction exists is $O(\frac{1}{n})$ for $k \geq 5\nu^{-1} \log_{u'} d'$ (Theorem 8.4). This proves our allocation strategy manages a swarm growth μ as long as the number of stripes is $c > \frac{2\mu^2-1}{u-1}$ and the number of copis per stripe is $k \geq 5\nu^{-1} \log_{u'} d'$.

8.2.3 Balanced heterogeneous systems

One of the main challenges in an heterogeneous system is to handle boxes with small upload capacity (i.e. lower than the stream rate $u_b < 1$). Let $u^* \geq 1$ be an *upload threshold* under which a box is considered to have deficient upload, i.e. less than u^*. We introduce the *upload deficit* with respect to u^* as the quantity $\Delta(u^*) = \sum_{b|u_b<u^*} u^* - u_b$ which is the overall bandwidth missing of *poor* boxes, i.e. boxes with upload capacity lower than u^*. A box b is considered *rich* when $u_b \geq u^*$. In this section, we are interested in heterogeneous systems satisfying:

$$u > 1 + \frac{\Delta(1)}{n}$$

Note that $u \geq 1 + \frac{\Delta(1)}{n}$ is an intuitive lower bound for scalability by considering the following request scenario. Suppose that all rich boxes request a video they do not possess and poor boxes start to play the same video v at maximum growth rate. Either v is widely replicated among the poor boxes, or rich boxes have to send data to poor ones because they have not enough capacity to exchange the video themselves. In the latter case, this requires an additional overall upload of roughly $\Delta(1)$.

We say that a system can be *u^*-upload-compensated* if for any poor box b we can reserve an upload capacity $u^* - u_b + 1 - u_b$ on a rich box $r(b)$ with $u_{r(b)} \geq u^* + (u^* + 1 - 2u_b)$. Several requests may fall in a box a as long as $u_a \geq u^* + \sum_{b|r(b)=a}(u^* + 1 - 2u_b)$. Note that this requires at least $u \geq u^* + \frac{\Delta(1)}{n}$.

Another challenge may come from the unbalance between storage capacity and upload capacity. Indeed, it may be useless to have very high storage capacity in boxes with low upload capacity and vice versa. A system is *u^*-storage-balanced* with respect to u^* if $2 \leq \frac{d_b}{u_b}$ and $\frac{d_b}{u_b} \leq \frac{d}{u^*}$ for all b. As a particular case, a proportionally heterogeneous system, where $\frac{u_b}{d_b} = \frac{u}{d}$ for all box b, is always u^*-storage-balanced for $d \geq 2$ and $u^* \leq u$. Note that a system with $2 \leq \frac{d_b}{u_b}$ for each

box b can always be considered as u^*-storage-balanced for $u^* \leq u$ by artificially reducing the storage capacity of each box to $d'_b = \tau u_b$ with $\tau = \min_b \frac{d_b}{u_b}$ at the cost of reducing the average storage capacity to τu.

We say that a video system is u^*-*balanced* if it is u^*-storage-balanced and can be u^*-upload-compensated. The main idea behind the requirement of compensated systems is to relay stripes for each *poor* box b (i.e. with $u_b < u^*$) through a rich box $r(b)$ with sufficient upload capacity according to the u^*-upload-compensated assumption. The request strategy is the following. A poor box b, which performs a video requests at time t, asks $r(b)$ to issue a request for its preloading stripe (selected as before): this is considered as a preloading request. At time $t+1$, $r(b)$ forwards this preloading stripe to b. This relies on the upload statically reserved on $r(b)$ and is not considered as a request. At time $t+2$, it requests c_b of the $c-1$ remaining stripes. At time $t+3$, it asks $r(b)$ to request the $c-1-c_b$ remaining stripes (these are postponed requests). At time $t+3$, $r(b)$ forwards these $c-1-c_b$ stripes to b (in addition to the preloading stripe). Again, these rely on the upload reserved on $r(b)$ and are not considered as requests. The strategy for a rich box a (i.e. $u_a \geq u^*$) whose user demands a video at time t remains similar except that the postponed requests are made at time $t+2$ instead of $t+1$. The request sequence is identical to the homogeneous case, except that the time round is scaled by a factor 2. As a consequence, the bound on swarm growth becomes μ^2 instead of μ.

It is now possible to extend Theorem 8.4 to balanced heterogeneous systems.

Theorem 8.8. *[136] For any fixed $u^* > 1$, consider a u^*-balanced (n,u,d)-video system. With high probability, a random permutation allocation with $c > \frac{4\mu^4}{u^*-1}$ and $k \geq 5\nu^{-1}\frac{\log d'}{\log u'}$ for $\nu = \frac{1}{c+2\mu^4-1} - \frac{1}{c+3\mu^4}$, $u' = \frac{c+3\mu^4}{c}$ and $d' = \max\{d, u^*, \exp(1)\}$ allows to successfully satisfy any sequence of requests with maximal swarm growth μ. For $c = \left\lceil \frac{10\mu^4}{u^*-1} \right\rceil$ and $u^* \leq 2$, it can achieve catalog size $\Omega\left(\frac{(u^*-1)^2 \log \frac{u^*+3}{4}}{\mu^4} \frac{dn}{\log d'}\right)$.*

The proof follows the same steps of Theorem 8.4. Lemma 8.5 can be generalized in this setting, and Lemma 8.7 stills holds. The rest of the proof of Theorem 8.4 can be immediately applied to this heterogeneous case.

8.3 Conclusion

In this chapter we have considered a fully peer-to-peer Video-on-Demand architecture, where a set of entities collaborate to store and distribute a catalog of videos. Taking into account the two main constraints of this kind of systems, the *network bandwidth* and the *storage capacity*, we have derived performance bounds in term of catalog size. We argue our results can also be applied to other architectures to derive the amount of contents they can store when resource allocation algorithms are constrained by storage and bandwidth limitations.

In a system where all peers have the same upload capacity and perform a video request, we have shown that there exists an average upload bandwidth threshold for enabling a scalable catalog of contents; this threshold is related to the average upload capacity of peers and is equal to the stream rate of the contents. Under that threshold, scalable catalog cannot be achieved, and

systems can store $m = \Omega(nc)$ contents. Above the threshold, linear catalog size is then possible and the problem of connecting nodes to serve demands reduces to a maximum flow problem. We have shown a similar threshold can be extended to heterogeneous systems as well.

Our results do not lead directly to a practical distributed algorithm but indicates the fully peer-to-peer approach is suitable to provide and on-demand streaming service. The design and the performance of practical resource allocation schemes for distributed on-demand streaming applications are presented in the following chapter.

Chapter 9

Practical algorithms for distributed Video-on-Demand applications

In this chapter we consider practical algorithms for distributed on-demand streaming systems. In particular, we focus on the *fully peer-to-peer* architecture for VoD applications. We suppose users access the service though their set-top boxes, which is a natural environment where this architecture may be deployed.

As explained in Chapter 7, there are two algorithms which are responsible for resource allocation: *video allocation* and *connection management*. *Video allocation* indicates how the videos of the catalog are allocated to boxes for storage. The service provider typically performs this allocation prior the actual use of the system[1]. *Connection management* is responsible for matching the boxes to allow them to download and play the desired videos.

In this chapter we focus on simple policies that can be used as building blocks of these algorithms. Our results give insights for the design and the parameter tuning of a peer-to-peer architecture for VoD. Nevertheless, they can be used in other applications, like UGC, or other architectures, like CDN or peer-assisted, where content allocation and connection management schemes are required. In particular, we consider the impact of: i) randomized/popularity based content allocation strategies; ii) the use of caching mechanisms for content distribution; iii) the use of static/dynamic policies in connection management algorithms.

Contents of this chapter are a joint work with Yacine Boufkhad, Didier Francey, Fabien Mathieu, Fabien de Montgolfier, and Laurent Viennot and are presented in [133] [38].

9.1 Algorithms

We now introduce the video allocation and connection management schemes that we are going to analyze in this chapter. We use the model presented in 8.2 whose parameters are reported in Table 8.2[2].

[1]The provider preforms the allocation in a centralized fashion; however, videos are stored in a decentralized manner at boxes.

[2]The only parameter not used here is the swarm growth μ. In fact, as explained in Section 9.2, we directly use realistic video request patterns.

9.1.1 Video Allocation

The video allocation cannot be based on a determined sequence of video requests because this sequence is not known in advance. However, video popularity may be inferred in several ways like: analysis of previous request sequences (for videos already proposed), customers' polls and so on. These techniques can estimate the number of requests for every video but cannot precisely predict what will be the exact video request sequence.

We focus only on the way videos are allocated and not on the mechanisms usedfor the actual distribution of videos from the content provider to boxes. As the allocation can be performed without rate requirements, we consider that it is not the most critical issue of the system; for instance, this may be done outside peak hours of a day or as background traffic. We consider two kinds of video allocation: *random permutation allocation* and *popularity based allocation*.

The **random permutation allocation (algorithm *U*)** does not take video popularity into account and all videos are replicated the same number of times. Then for a given catalog size m we have for all videos $k_v = k$, with $k = \lfloor \frac{dn}{m} \rfloor$. As presented in Section 8.2.1, the system generates a random ordered list of the ckm stripe copies to be stored, where each stripe appears k times exactly, and allocates these stripes to the cdn stripe storage slots of the n boxes. On the other hand, in the **popularity based allocation (algorithm *P*)**, k_v is computed according to the video popularity, i.e. every video v is replicated proportionally to the expected number of requests. A minimum number of copies of a video (for instance 1) is however guaranteed.

9.1.2 Connection management

The connection management algorithm performs the on-line matching between "server" boxes storing video stripes and the "client" boxes downloading videos. In our model a client that performs a video request, receives, for every stripe of the desired video, a list of "server" clients that can potentially upload the stripe; in order to avoid too much overhead, the size list is bounded by a maximal value x (if more than x boxes can potentially provide the stripe, only a random subset of size x will be used). We do not consider here the way a client obtains this list, but for example it can be obtained from a central tracker (solution implemented in our prototype Section 9.3) or a DHT.

Once the list is received, the client box contacts the potential server boxes starting from the one with the greatest remaining upload capacity, until it finds one box that accepts to upload the stripe to it. The remaining upload capacity can be discovered by exchange of messages between the client box and the server boxes, or it can be included as additional information in the box list. This mechanism is intended to balance connections over boxes.

A given server box can accept up to $\lfloor cu \rfloor$ simultaneous stripe connections and an acceptance policy must be defined.

We propose three possible algorithms to populate the box list and to define the acceptance policy of server boxes.

Storage Only (algorithm S) This is the simplest algorithm where the box list is populated only with peers storing the video stripe from the video allocation. $\min(x, k_v)$ boxes are randomly selected between the k_v ones owning the stripe. A server box accepts the incoming stripe requests if and only if it has enough remaining upload capacity.

Caching and Relaying (algorithm C) This strategy is based on the fact that while a video is being watched, it is also *cached* within the storage device of the box. The set of boxes watching a given video v is called the *swarm* of v. On a new request, the box list is populated by boxes *storing* the video stripe from the video allocation and by boxes from the swarm. The choice is uniform at random in the union of these two sets. A server box accepts all stripe requests it receives while it has enough upload capacity, otherwise it refuses.

Dynamic Relaying (algorithm D)

The dynamic relaying is similar to Algorithm C. It only differs in the stripe request acceptance policy: when a given server box with no available upload slots (i.e. handling $\lfloor cu \rfloor$ simultaneous stripe connections) receives a request for a stripe *stored in its cache*, it selects a connection serving a stripe of the *video allocation* (if any), discards it and accepts the incoming request instead. The box whose connection has been discarded in the process has to look for another box to establish a new stripe connection.

The idea behind this algorithm is to handle very popular videos. If the demand for a given video is very high, it may become necessary to give priority to additional replicas of the video (i.e. to cache connections), over video allocation connections.

The video allocation and connection management algorithms considered in this chapter are summarized in Table 9.1.

9.2 Simulative analysis

To analyze the performance of a fully peer-to-peer VoD system we have developed an event-based simulator. The simulator first allocates videos to boxes according to the video allocation algorithm and then simulates a video request process. For each video request, the connection management algorithm is run. Note that for schemes running algorithm D some boxes may suffer from disconnections for some stripes. These boxes should then perform stripe requests for the disconnected stripes.

We define the *startup time* (T_S) as the maximal delay needed by a box to contact the boxes in the box lists and to establish c connections to download the c stripes of the desired video. We thus tune the granularity of our simulator to T_S. This is a conservative assumption for the boxes suffering of one or few stripe disconnections, as those boxes may actually need less than T_S to recover from these disconnections.

Once a box is connected, its cache is filled according to content and bandwidth availability of server peers (we conservatively assume that all caches are initially empty). When there are enough data in its cache (B_S), a box starts playing out the video at SR. When a box ends the download of a stripe it releases upload resources of the server peer and when it finishes video play out the box is ready to perform a new video request. A *failure* occurs when no data are available for the play out, i.e. the cache is exhausted.

9.2.1 Simulation set up

Applications should be designed and tuned to be robust against the worst working conditions. In particular, they should be able to face peak hours of the day, with maximal service demand,

Notation	Schemes
SU	Storage only / Random permutation allocation
CU	Caching and Relaying/ Random permutation allocation
SP Err	Storage only / Popularity based allocation *Err* is the error between estimated and actual requests
CP Err	Caching and Relaying/ Popularity based allocation *Err* is the error between estimated and actual requests
DU	Dynamic Relaying / Random permutation allocation

Table 9.1 : Algorithms analyzed in this chapter.

and flash-crowd video requests. So, we evaluate our schemes under such specific scenarios.

Unless otherwise stated we suppose there are $n = 1000$ boxes in the system; this is a typical population size for systems deployed by an ISP within a last mile subnetwork (for instance DSL users depending on the same DSLAM) where the VoD system is the most useful, since it does not load the network core. We set $T_S = 5$ seconds as a conservative start up value, $B_S = SR{\cdot}T_S$ and we suppose boxes have homogeneous upload $u = 1.2SR$ and storage $d = 25$ capacities.

The n boxes generate a video request process where the number of arrivals per T_S follows a modified Poisson distribution. This distribution has been observed in [125] where real traces of a VoD system are analyzed. During peak hours, the maximal and the mean number of requests over a period of $5s$ (equals to T_S) are $\delta_{max} = 27$ and $\delta_{mean} = 17$ respectively. The maximal list size x is set to 30 and we suppose every box can perform one video request.

We suppose that the video popularity *estimated* by the content provider follows a power law distribution of slope $\alpha = 1.4$. Note that this distribution is more skewed than the one observed in [125] for VoD video popularity but allow us to take into account more skewed popularity distributions like the ones observed in a UGC service analysis [19].

In order to simulate a certain inaccuracy in popularity prediction, we assume that the *real* video request process follows a *noisy* popularity distribution. This noise is obtained by using a permutation of the video ranking so that the normalized precedence distance between the real and estimated rankings reaches a certain percentage. Unless otherwise stated we suppose the inaccuracy is of 20% (denoted as P20 on figures).

Videos are split in $c = 10$ stripes by default, and are supposed to be of infinite length since we focus on a flashcrowd scenario where all video requests are performed over a period shorter than video duration. As in the previous chapter, for simplicity we normalize $SR = 1$.

9.2.2 Performance metrics

The main performance metrics are the catalog size m and the failure tolerance ϵ. ϵ is the maximal ratio of failed downloads that may occur. These two metrics are correlated: for a given ϵ, the maximal sustainable catalog size is $m = \lfloor \frac{dn}{k} \rfloor$ for the minimal value of k allowing the system to have less than ϵn video requests not fulfilled. Unless otherwise stated, the failure tolerance is set to 1%.

To handle download failures a service provider may use backup mechanisms: for instance a box that fails to download a video from other boxes, may contact a dedicated server. In such a way

the central entity has only to serve a few percentage (i.e. the failure tolerance) of video requests.

9.2.3 Results

As stated in the previous chapter, the catalog size m, which is in inverse proportion to the redundancy k, is closely related to the upload provisioning u. We therefore focus first on the relation between the two performance metrics m and ϵ, and the upload provisioning u.

For a given system, if two out of m, ϵ, u variables are known (the other parameters being fixed), the last one will be a consequence of the first two. A given failure tolerance and a given upload bandwidth will determine the sustainable catalog size; the failure tolerance can be computed for a given catalog size and upload provisioning; it is possible to deduce the minimal upload provisioning needed to store a catalog of a given size and to respect a given failure tolerance.

All parameters but m, ϵ and u being fixed, we define a trade-off space as follow: we say a triplet (m, ϵ, u) is *optimal* if the system works with these parameters, but fails if we choose $m' > m$, $\epsilon' < \epsilon$ or $u' < u$. The trade-off space is then defined as the set of the optimal triplets.

For better readability, we display three slices of the trade-off space for the five schemes presented in table 9.1. In each plot, the optimal value is computed by averaging multiple simulation runs.

Bandwidth and catalog size

Figure 9.1 represents the achievable catalog size m as a function of the upload provisioning u for a fixed failure tolerance $\epsilon = 1\%$.

First, we observe that SU and SP20 schemes (based on a *storage only* connection management) perform poorly with respect to the three other schemes, and take little advantage of extra available bandwidth. The reason is that those schemes cannot serve more than cuk_i times a given video v: a given stripe is replicated in k_v boxes, each box being able to upload it at most cu times. Popular videos are requested a lot of times, so that lot of replicas of those videos are needed to respect the failure tolerance ϵ. Due to inaccuracy of the popularity estimation, SP20 does not perform better than the agnostic uniform allocation. The impact of the popularity estimation inaccuracy will be detailed in Section 9.2.4.

On the other hand, there is an important catalog size improvement for the three other schemes when the upload capacity increases. For $u = 1.5$ (that means 50% more bandwidth than needed for the feasibility of the system), all three schemes achieve a catalog size equal or close to $nd = 25000$, which is the maximal possible size given the physical storage capacities, i.e. one copy per video $k = 1$. In details, CP20 can store less videos than CU and DU. DU outperforms CU for small bandwidth over-provisionning (up to $u \approx 1.2$), where critical situations requiring re-connections are most likely to happen; for larger values of u, both schemes perform similar.

Failures and catalog size

Figure 9.2 shows the achievable catalog size m as a function of the failure tolerance ϵ for a given upload provisioning $u = 1.2$. The performance order of the schemes is the same as already observed in Figure 9.1.

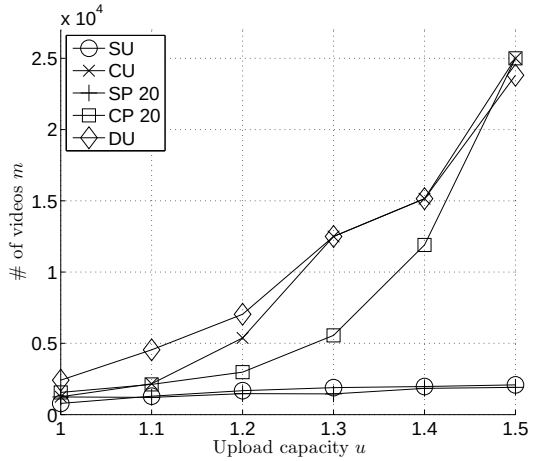


Figure 9.1 : Catalog size m as a function of the upload provisioning u ($\epsilon = 1\%$).

SU and SP20 achieve the smallest catalog sizes, with little variation with respect to ϵ. In other words those schemes suffer from a threshold effect: around a certain critical redundancy $k \approx 20$ (corresponding to a catalog size of about 1300 videos) failures strongly increase so that redundancy should be set around this critical value to respect the failure tolerance. As stated before, the weak point of cacheless schemes is popular videos. As they represent a significant part of the requests, if there is not enough redundancy to serve them a significant amount of failures should be expected.

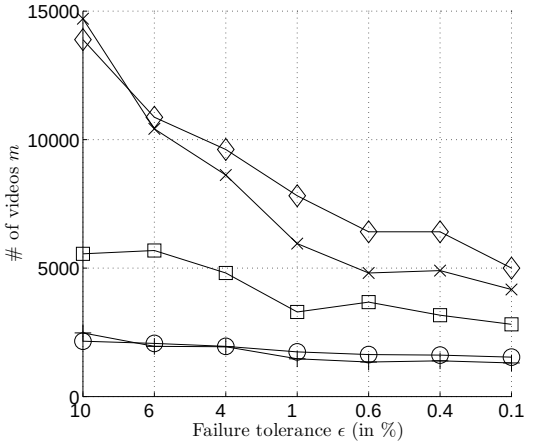


Figure 9.2 : Catalog size m as a function of the failure tolerance ϵ ($u = 1.2$).

Among the three other schemes CU seems to be the most sensitive to the failure tolerance, with a ratio 3 between the smallest ($\epsilon = 0.1\%$, which states that no more than one failure over the $n = 1000$ requests) and largest ($\epsilon = 10\%$) considered tolerances. DU outperforms the other schemes for small value of failure tolerance.

Bandwidth and failures

Lastly, Figure 9.3 shows how, for a given catalog size of $m = 5000$ videos, the upload over-provisioning u helps to decrease failures. The considered catalog size is largely greater than the one sustainable for SU and SP20; we thus observe without surprise that those schemes generate an important amount of failures whatever the bandwidth is. The other schemes can actually take advantage of the over-provisioning to significantly reduce ϵ, DU being the most efficient for this catalog size, followed by CU and CP20.

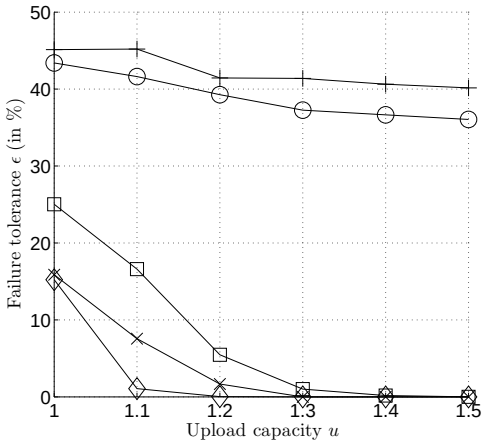


Figure 9.3 : Failure tolerance ϵ as a function of the upload over-provisioning u ($m = 5000$).

9.2.4 Impact of the video request distribution

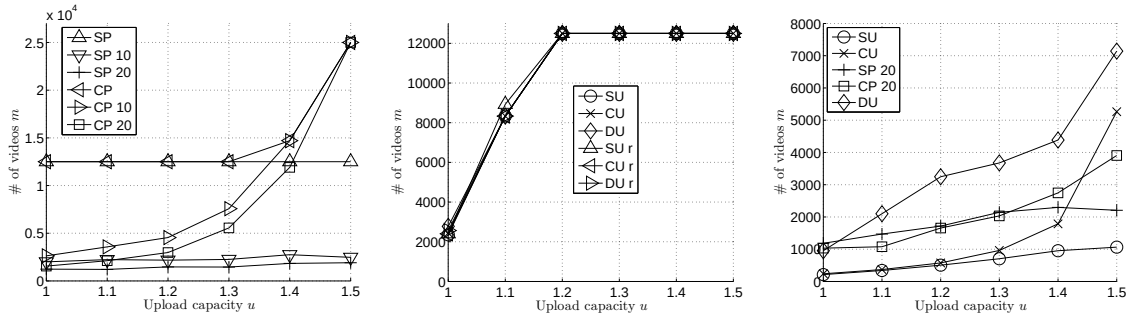
In this section we analyze the impact different video request patterns have on performance.

First we focus on the accuracy of prediction for *popularity-based* allocation. In particular we analyze, in Figure 9.4(a), SP and CP when the video request process either follows exactly video popularity, or is 10% or 20% inaccurate.

If popularity prediction is accurate, both SP and CP can store a large number of videos. In particular CP can store larger catalogs than SP thanks to caching for $u > 1.3$. However, performance dramatically decreases as soon as popularity prediction is inaccurate. SP requires a large number of copies starting from an inaccuracy of just 10%. On the other hand CP can store catalog of sizes comparable to the accurate case only for large values of u.

This highlights the fact that *popularity-based allocation* is not suitable because it is very sensitive to popularity prediction. Moreover, as presented in the previous section, *random permutation allocation* outperforms *popularity based allocation* while being easier to deploy and unaffected by prediction inaccuracy.

Figure 9.4(b) reports performance of the schemes based on *random permutation allocation* for random uniform video requests, and for video requests following a power law with a slope of $\alpha = 0.2$. This last video popularity has been observed in [125] and it is less skewed than the one used for the rest of the analysis.



(a) (b) (c)

Figure 9.4 : Catalog size for different popularity prediction accuracy (a), different video request process (b), and for one popular video (c) as a function of the upload over-provisioning.

First, we observe that schemes behave similarly in both models. This is because the skew factor is not big enough to observe any difference with respect to the uniform distribution. In these scenarios, all schemes achieve almost the same catalog size for all u values and for $u \geq 1.3$ two copies per video are needed to obtain a failure tolerance $\epsilon < 1\%$.

We can conclude that for a uniform or slightly skewed video request process all schemes based on *random uniform video allocation* behave similarly, independently of the connection management algorithm. This is because almost all video requests are satisfied by the video allocation, so that caches and dynamic relaying are not exploited.

If we compare figure 9.4(b) to figure 9.1 it is possible to notice that catalog sizes are larger for uniform (or lightly skewed) video request process for $u < 1.3$. On the contrary a larger slope in video request process allows schemes to reach the maximum catalog size ($nd = 25000$) for $u = 1.5$, while for uniform video requests this is not attained for $u \leq 1.5$. This highlights a more skewed video popularity requires larger amount of bandwidth, but can provide larger catalogs of videos if the system is sufficiently over-provisioned.

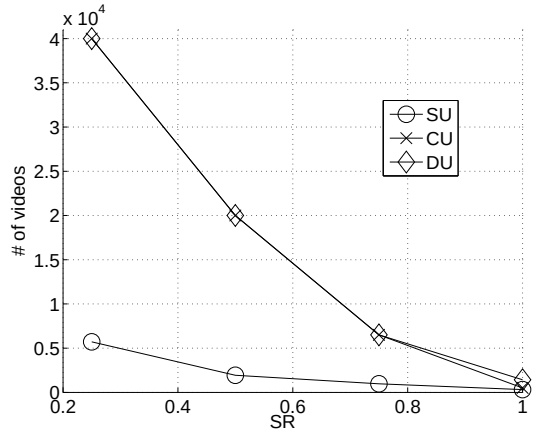
Finally, we analyze an extreme scenario where half of the boxes ask for videos according to the noisy video popularity distribution, while the remaining 50% ask for the same video. This scenario can represent for example the release of a very popular video that many users want to view immediately.

We observe in Figure 9.4(c) that the catalog size is much smaller than in the reference scenario (Figure 9.1) because of the very skewed video request pattern. In particular, DU clearly outperforms the other algorithms while CU is more affected by this video request pattern. This highlights that the use of cache is not enough to support such kind of scenarios while it is necessary to give priority to the cached copies of the very popular video as DU does.

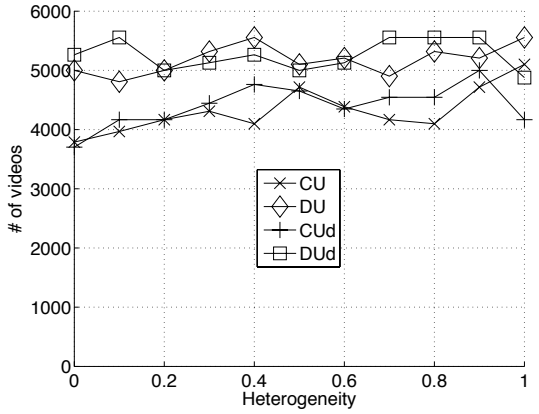
9.2.5 Impact of the other parameters

We now briefly describe the influence of the other system parameters not discussed yet.

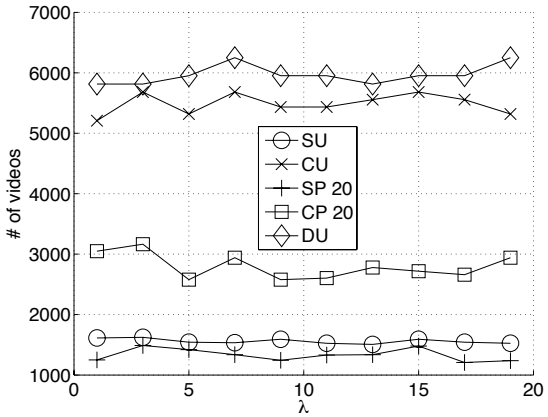
SR is probably the most important parameter to tune in a given physical system, because it affects both $d = \frac{\mathcal{D}}{T \cdot SR}$ and $u = \frac{\mathcal{U}}{SR}$. For given physical capacities, increasing SR can improve the quality of videos at the price of lowering both the logical capacity and the relative bandwidth over-provisioning of the system. The influence of SR is shown in Figure 9.5(a), for a system



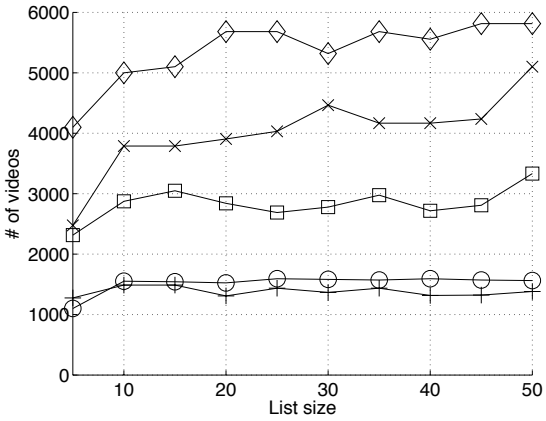
(a) $m(SR)$



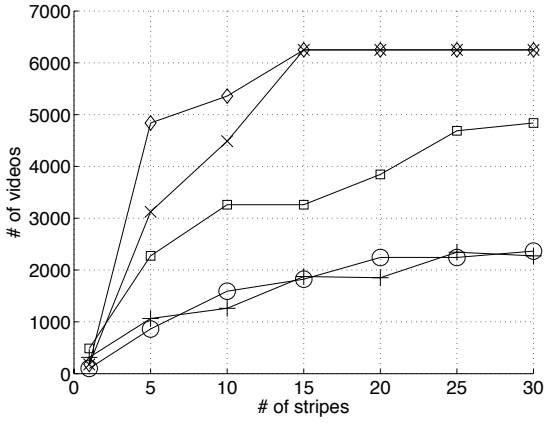
(b) $m(h)$: d suffix indicates storage capacities are proportional to upload capacities



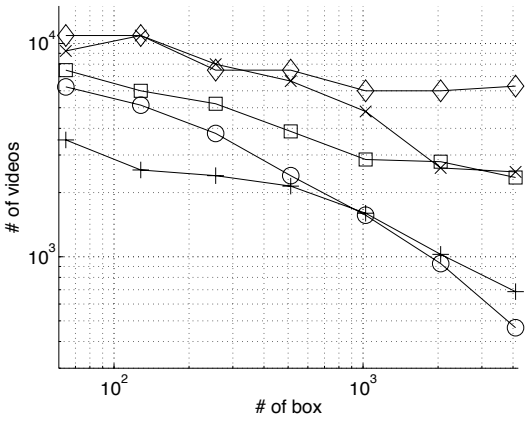
(c) $m(\lambda)$



(d) $m(x)$



(e) $m(c)$



(f) $m(n)$

Figure 9.5 : Catalog size m as a function of the video rate SR, the upload capacity heterogeneity h, the arrival intensity λ, the size of the box list x, the number of stripes c, and the number of boxes n.

where the physical storage capacity is $10T\mathcal{U}$. For $SR = \mathcal{U}$ ($d = 10$, $u = 1$) the performance is poor as expected (m is less than 1500). To decrease video rate to $SR = 3/4\mathcal{U}$ ($d = 13.33$, $u = 1.33$) increases the catalog size to 6500 videos for the best schemes (CU and DU). If $SR = \mathcal{U}/2$ ($d = 20$, $u = 2$) the system will admit a catalog of 20000 videos, instead. From this point, the bandwidth overprovisioning is high enough to allow the system to work without redundancy, and the catalog size is $m = nd = n\frac{\mathcal{D}}{T \cdot SR}$ (the additional gain is only due to the increase of d).

The other parameters have less influence on system performance, and most of the time, it suffices to have them *big enough* or *small enough* to consider them tuned.

- *Heterogeneous upload capacities*: in the previous chapter we propose to set the storage capacity of every box proportional to its upload capacity to deal with heterogeneity. In figure 9.5(b) we investigate the impact of heterogeneity for proportional (indicated with a suffix d in figure) and constant storage capacity. We define as h the heterogeneity parameter so that hN boxes have an upload capacity randomly chosen between $u = 0.6SR$ and $u = 1.8SR$ while the remaining $(1 - h)N$ have $u = 1.2SR$. We observe heterogeneous upload capacities do not affect the catalog size schemes can achieve for both proportional and constant d.

- *Arrival intensity:* contrary to all other parameters, the arrival intensity depends on the user behavior and can hardly by tuned by the service provider. We consider here the video request process follows a Poisson distribution with average number of arrival per second equal to λ. Figure 9.5(c) highlights that, as long as the intensity stays within a reasonable range (less than 100 arrivals per second), it has no effect on performance. Of course, there is a threshold, and a too large flashcrowd (for instance the n requests being launched simultaneously) definitively overwhelms the system. However, realistic intensity values like the ones observed in [125] are far below that threshold, so intensity does not seem to be a key issue in practice.

- *Size of the box list:* increasing the list size x improves the chances for a client to find boxes able to upload the stripes it needs. Our simulations indicate that most of the performance is reached before the value $x = 10$, and it slowly grows, or stagnates, after that (cf Figure 9.5(d)). Considering that the connection management overhead is proportional to x (a client must manage x potential servers per stripe), the small performance gain obtained by using large values for x may not be interesting.

- *Number of stripes:* increasing c provides a regular improvement up to $c = 30$ for the schemes SU, SP and CP (cf Figure 9.5(e)). For CU and DU, optimal value is reached at $c = 15$. Overhead containment suggests not to use larger values of c.

- *Number of boxes:* in figure 9.5(f) we vary the number of boxes n while keeping constant the total storage capacity of the system, i.e. nd constant. We observe for all schemes that the catalog size shrinks as the number of boxes increases. CU and DU behave similarly for small n, while DU outperforms CU for a large number of boxes. This highlights that dynamic schemes are suitable when the box population is large.

9.3 Experimental evaluation

A working prototype of an on-demand streaming system has been developed by Didier Francey in 2009 during its internship at Orange Labs; a report on this activity, as well as a detailed description of the software can be found in [38].

The prototype is coded in Java and allows to set-up different architectures for the distribution of the streaming content. It is therefore possible to evaluate the performance of the different approaches described in Chapter 7.

The software is organized in three main building blocks: the **provider**, the **storage** and the **client** modules. The software can be run by using one of the three modules only, or more than one concurrently.

The *provider* module is the centralized element of the system. It acts as a service provider and is responsible to manage the content allocation for the storage, and to provide information about the content locations to the clients. It is also responsible to collect information from storage sites about their current status: number of clients connected, remaining upload capacity and so on.

The *storage* module is responsible for the storage of part of the content catalog as indicated by the provider module. The *client* module is instead responsible to perform video requests to the system by contacting the provider. Server module periodically sends messages to the provider to update it about its current status.

To realize a fully peer-to-peer architecture it is sufficient to run one instance of the software with the provider module active, and several other instances with the storage and client module concurrently active. To set-up a peer-assisted architecture there is the need of one provider, one storage entity and several clients with caching and upload capabilities. Finally for a CDN, clients and storage sites will run separately on different instances of the software.

As concern *transport protocols*, the prototype implements both TCP and UDP capabilities. We argue that, for control messages UDP is the natural choice, while for data transfers both TCP and UDP can be used. Control messages are few tens of bit in size and do not require a reliable transport protocol because they are periodically retransmitted. On the other hand, data require reliable transfers and are transmitted at a constant rate. TCP seems the more natural choice because it offers this reliability, so that no other mechanisms have to be implemented at application level. However, the use of UDP would eventually reduce the transmission delay but it would require additional reliability mechanisms in the software.

9.3.1 Performance evaluation

We validate the results presented in Section 9.2 for a peer-to-peer VoD system by running experiments over the Grid5000 platform [93]. This platform has already been used for the PULSE experimental evaluation and is presented in Section 4.3. In order to perform our experiments, we artificially limit the upload capacity of the boxes by implementing a configurable upload bandwidth cap in the software.

We do not report here all the results obtained but just one significant plot comparing the performance obtained during the experimental evaluation with the simulative results. In Figure 9.6 we observe experimental results confirm the simulative evaluation presented in the previous

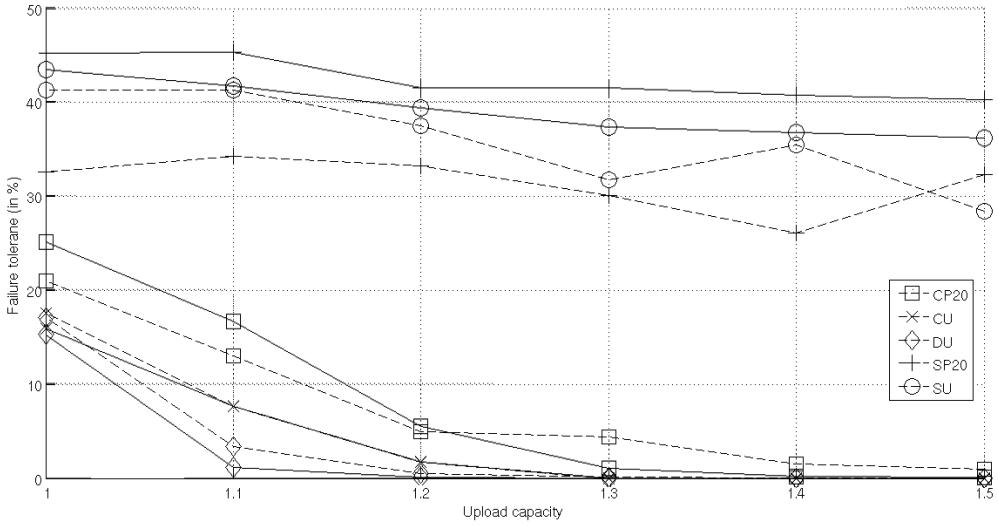


Figure 9.6 : Comparison between experimental (dotted line) and simulative results. Failure tolerance as a function of the upload capacity.

section. The only significant difference is noticeable for the *Storage only / Popularity based allocation* scheme. This policy has in fact been optimized in the prototype in order to avoid storing the same stripe twice in the same box.

For more details about the prototype, as well as the complete set of experimental results, please refer to [38].

9.4 Conclusion

In this chapter we have considered simple storage and connection management policies for distributed on-demand streaming systems. By means of extensive simulations we have analyzed the *catalog size-upload provisioning-failure tolerance* trade-off, the impact of different video request patterns, and the role of system parameters. Our simulative results have been confirmed by an experimental evaluation performed with a software prototype we designed and developed.

We have shown video allocation policies based on *video popularity* are not suitable because they are too sensitive to prediction accuracy. Moreover, simple *random* video allocation performs well while being easy to deploy and robust against video request distribution. On the contrary, the use of cache to allow nodes to distribute the video they are currently downloading is critical to improve system performance.

Dynamic connection algorithms are not crucial for common arrival and video request patterns while they are suitable in extreme scenarios. We believe such kind of extreme scenarios, with high churn of requests for just one or few videos or slightly over-provisioned "servers", are not that rare in practice. Moreover, it seems that dynamic connection algorithms outperform static ones when the system size is large.

To over-provision the upload capacity can help to increase the catalog size and to reduce the failure tolerance. However, to increase the upload provisioning is expensive in term of network

resources, unless a virtual increase is made by lowering the video stream rate (but video quality is affected in that case). On the other hand, to increase the failure tolerance will increase the catalog size and/or reduce the required upload provisioning. A higher failure tolerance will reduce QoS perceived by users too, unless failures can be recovered in some way (for example by the use of a backup server i.e. hybrid architecture). We believe the use of special coding mechanisms may improve the performance of the considered schemes; in this chapter we have not considered the role of coding, and we leave it as subject of future work.

The other system parameters are not that crucial even if some default values (as for number of stripes or box list size) are recommended.

Chapter 10

Conclusion of PART II

For several years, cable and satellite providers, and video rental companies, have been developing on-demand streaming services for an increasing population of users. Later, ISPs, like Orange, have also started to provide content on-demand to their customers through set-top boxes; these early services all rely on a centralized architecture.

A big shift for on-demand streaming is represented by the advent of User Generated Content applications (UGC), like YouTube or DailyMotion, where users can upload their own media files that are then available to all the other users of the system. Moreover, the increasing popularity of catch-up TV services proposed by several TV broadcasters, coupled with a growing interest in traditional VoD, is increasing the the traffic generated by on-demand streaming, that represents today the largest part of the Internet traffic, and is expected to double every two years towards 2013 [36].

To deal with this increasing audience, several decentralized architectures have been proposed for both customers and content catalog scalability. For instance, YouTube exploits the Akamai CDN for content delivery, while PPLive VoD service is based on a peer-assisted architecture.

In this chapter we have considered the *fully peer-to-peer* approach to on-demand streaming, where customers of the service actively collaborate to store the content catalog and serve requests generated by the users. This architecture can for example be deployed on set-top boxes installed at user home that have both storage and network capabilities. Of particular interest is the scenario where the service is deployed at set-top boxes within the same last-mile subnetwork (for instance DSL users connected to the same DSLAM), because the traffic will remain local and won't traverse the core of the network.

We have first considered the size of the content catalog these applications can provide to users by taking the bandwidth and storage capacity constraints into account. We have shown that there exists a bandwidth threshold corresponding to the stream rate of contents that is decisive for the catalog scalability. Note that this threshold corresponds to the required bandwidth for the feasibility of the system according to the analysis performed in Chapter 2. If the average capacity of system entities is lower that this threshold catalog scalability cannot be achieved. On the contrary, if it is larger than this threshold we have shown it is possible to store catalog of size linear in the number of users. These results are valid for homogeneous systems, where users have all the same upload and storage capacity, but can also be extended to heterogeneous environment if a balancing mechanism is employed.

Then, we have analyzed practical policies for the design of content allocation and connection management algorithms. The former is responsible of the distributed storage of contents, while the latter is responsible of the matching between users storing a given content and the user requesting that content. We have shown popularity-based allocation mechanisms are not suitable because too sensitive to popularity prediction, while random allocation mechanisms are very efficient if coupled with caching mechanisms to redistribute the contents that are currently watched by users. We have shown dynamic connection management algorithms are only needed in extreme scenarios where the number of users is large, the upload capacity is slightly over-provisioned and there are only few very popular contents. We argue these scenarios are not that rare in practice.

There are some open questions and problems related to on-demand streaming that are current or future interesting research topics:

- *Experimental comparison of the different architectures for on-demand streaming.* We have shown, by means of test-bed evaluation, a fully peer-to-peer architecture is feasible and efficient to set-up an on-demand streaming applications. The experimental evaluation we have performed has confirmed the theoretical and simulative results we have derived. However, a comparative evaluation of the different architectures (peer-to-peer, peer-assisted and CDN) in a wide range of scenarios is still missing.

- *Unstructured mesh-based content distribution techniques.* The content distribution mechanisms we have considered are highly structured and tailored to controlled environment. We believe a mesh-approach for content distribution would lead to more efficient and resilient distribution techniques. First efforts in this direction have been proposed without taking into account the content allocation. We believe the design and the analysis of a mesh-approach coupled with content allocation is a good track for future research.

- *Stream coding.* In this chapter, we have considered basic coding mechanisms only (the striping), but enhanced source or network coding techniques may improve performance and impact the design choices of systems.

- *Efficient content catalog update mechanisms.* To the best of our knowledge, there are no previous works that consider the way multimedia files are transferred from the content provider to the storage entities. A common way to circumvent the issue is to upload them during night or as background traffic. This is possible if catalog updates are performed every so often (e.g. once a week) as it is in common VoD services. However, this is no more possible if we consider UGC applications where users upload thousand of new files every day.

Chapter 11

Conclusion

The peer-to-peer architecture represents an enormous potential for the deployment of Internet services and applications. Because of its inherent scalability, the resources of a system based on a peer-to-peer approach increase with the number of participants, both ISPs and end-users can take advantage from this. ISPs can provide a large set of new services with very limited costs, and perhaps exploit devices they already have, like for instance set-top boxes and gateways installed at home. End-users, on their side, can set up applications at very low cost by simply relying on the resources of other users interested in the considered service.

The drawbacks of a P2P solution are of technical and legal nature. Technical issues are related to the decentralized structure of P2P: this introduces additional challenges for the management and allocation of the distributed resources, in order to meet application requirements. As concern legal issues, the control on a P2P based service is very limited for both juridical entities and ISPs, particularly if the application runs on user PCs. This may lead to the deployment of illegal activities at low costs. In this thesis we only consider technical aspects of the problem.

The last decade has seen the deployment of a large spectrum of P2P applications such as file-sharing, telephony, storage, gaming and so on. In this thesis we have considered the diffusion of multimedia streaming. In particular, we have focused to mesh based live streaming systems designed for uncontrolled environments, and to Video-on-Demand streaming applications tailored for devices under the control of an ISP.

Contributions

Before to consider a specific streaming application, we have quantified the benefits that a P2P based architecture may provide in term of network bandwidth. We have highlighted that a provider can reduce its costs while largely increase its network capacity. We have derived conditions for the feasibility of on-demand and live streaming systems, as well as file-sharing applications, and we have analyzed the number of customers a P2P architecture can potentially serve.

In the first part of the thesis, we have considered the distribution of live streaming in uncontrolled environments. The most popular commercial applications are based on a mesh approach, and can handle a large population of users. We have shown the mesh approach can also achieve

optimal diffusion performance, and we have demonstrated the existence of a diffusion rate/delay trade-off in several practical diffusion schemes.

We have shown that a mesh approach coupled with incentive mechanisms is able to meet the live streaming requirements and can be used for the deployment of a real application over the Internet. This solution is effective to incentive peers to resource sharing by providing better media quality and/or lower reception delay to the nodes that contribute the more. We have shown that TFT-like diffusion schemes can perform as bandwidth-aware algorithms while being much easier to be implemented in real networks. We have highlighted that an equilibrium between resource awareness-agnosticism is needed to improve the system performance, and that a rate/delay trade-off arises as a function of the level of awareness/agnosticism. Finally, we have shown that a wrong tuning of system parameters may strongly affect diffusion performance.

In the second part of the thesis we have considered the case of on-demand streaming. We have analyzed the size of the content catalog a fully peer-to-peer approach can provide to users as a function of storage and bandwidth constraints. We have shown that a catalog whose size is linear with the number of users can be stored as soon as the system is slightly bandwidth over-provisioned. We have considered simple content allocation and distribution techniques that can be used for the design of an on-demand P2P application. We have highlighted that random content replication coupled with caching at users is efficient, particularly if dynamic connection management algorithms are employed.

Outlook

Peer-to-peer traffic has kept increasing in last years, and is now still increasing in volume but declining as a percentage of total IP traffic. This is due to the very high percentage of video streaming data that is increasing faster than the P2P one. This traffic is mainly related to UGC applications, which nowadays rely on client-server or CDNs architectures.

Streaming applications can largely benefit from the use of a peer-to-peer approach; not only the already popular live streaming but also on-demand services, like UGC, can reduce their costs by using a P2P architecture. To face this increasing audience in next years more and more providers may probably move toward a P2P based solution.

Peer-to-peer architectures are also very effective if employed by ISPs in controlled scenarios, like for example to provide on-demand services through set-top boxes. We believe these solutions are very attractive for ISPs because the required hardware is already available or will be available in near future. In next years we will probably assist to the deployment of several services, like VoD, live streaming and gaming, based on these controlled peer-to-peer solutions.

Moreover, ISPs would probably be interested in deploying P2P software to provide services to customers when they are not directly connected to their networks. Consider people that are outside their countries for short or long term periods and would like to watch their national TV channels; a P2P architecture is an interesting solution to provide them this service at very low costs. For instance, this business model is already used by PPLive that allows chinese people to watch their national channels everywhere in the world.

ISPs are also showing increasing interest in collaborative solutions between operators and P2P networks. Standardization efforts, like ALTO [120], or research projects, like Napa-Wine [79], are testimonies of these trends.

Considering these evolutionary trends, the deployment of peer-to-peer based architectures for media streaming in both controlled and uncontrolled environments is probably one of the major challenge of the near future. This is also confirmed by the existence of several projects devoted to this topic, like P2PIm@ge [84], Napa-Wine [79], P2P-Next [83] and so on.

We believe live streaming applications achieving near-optimal diffusion performance may be deployed by using a mesh-incentive approach and epidemics-style distribution algorithms. These solutions may easily integrate network and locality aware mechanisms. Further analysis is required to improve these network awareness solutions, and the interaction between resource allocation at network and application level is not completely understood yet. Moreover, lot of P2P applications for live streaming are designed for the diffusion of a single stream; additional research is needed to allow the stream of multiple channels and fast channel zapping [56].

As concern on-demand streaming, in our opinion random content allocation and caching mechanisms are effective to provide the service. However, the connection techniques we have proposed are highly structured and may suffer in dynamic environments. Some mesh-based solutions have been proposed (for instance [7]), but they do not take content allocation into account. We think further studies are required to couple random-caching allocation techniques with mesh-based diffusion mechanisms.

We believe efficient mechanisms for proactive distribution of the contents to store in on-demand applications may become worthwhile. In fact, this proactive distribution is not an issue if the target application is a VoD service where contents are updated every so often. However, this is a challenge in a UGC application where fast and efficient catalog update techniques are needed. In our opinion, these mechanisms should definitely be investigated.

The increasing number of overlay networks and the continuous emergence of new applications, clearly highlight how the interest of users is nowadays focused to contents and services instead of the actual entities storing them. The current network infrastructure is however still designed around the end-to-end approach. This situation has lead to an increasing popularity of the Content-Centric Networking concept, recently promoted by Van Jacobson [112]. Under this approach, the network focuses on data instead of physical location of objects. Considering the evolutionary trends, this solution is very appealing for the development of future networks, and both users and service providers can benefit from it. Some of the insights derived in this thesis, particularly the ones related to caching and storage techniques in on-demand streaming, may be useful for the on-going research on Content-Centric Networking for the caching and forwarding of data objects.

Bibliography

[1] Internet world stats. http://www.internetworldstats.com/dsl.htm.

[2] `http://iblnews.com/story.php?id=17429`.

[3] L. Abeni, C. Kiraly, and R. Lo Cigno. On the optimal scheduling of streaming applications in unstructured meshes. In *Proc. of Networking*, 2009.

[4] S. Agarwal, J. P. Singh, A. Mavlankar, P. Baccichet, and B. Girod. Performance of P2P Live Video Streaming Systems on a Controlled Test-bed. In *Proc. of International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*, 2008.

[5] S. Ali, A. Mathur, and H. Zhang. Measurement of commercial Peer-to-Peer live video streaming. In *Proc. of Workshop in recent advances in Peer-to-Peer streaming*, 2006.

[6] M. S. Allen, B. Y. Zhao, and R. Wolski. Deploying video-on-demand services on cable networks. In *Proc. of the 27th Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 63–71, Washington, DC, USA, 2007. IEEE Computer Society.

[7] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez. Exploring VoD in P2P swarming systems. In *Proc. of IEEE INFOCOM*, pages 2571–2575, 2007.

[8] C. H. Ashwin, R. Bharambe, and V. N. Padmanabhan. Analyzing and improving a bittorrent network performance mechanisms. In *Proc. of IEEE INFOCOM*, 2006.

[9] F. Baccelli, D. Hong, and Z. Liu. Fixed point methods for the simulation of the sharing of a local loop by a large number of interacting TCP connections. In *Proc. ITC Specialist Conference on Local Loop, Barcelona, Spain*, 2001.

[10] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM*, 2002.

[11] S. Banerjee, T. Griffin, and M. Pias. The interdomain connectivity of planetlab nodes. In *Proc. of PAM*, 2004.

[12] M. Bawa, H. Deshpande, and H. Garcia-Molina. Transience of peers and streaming media. In *Proc. of HotNets-I*, 2002.

[13] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, S. Seshan, and H. Zhang. The impact of heterogeneous bandwidth constraints on dht-based multicast protocols. In *Proc. of 4th International Workshop on Peer-to-Peer Systems*, 2005.

[14] BitTorrent. http://www.bittorrent.com/.

[15] R. Blahut. Theory and practice of error control codes. In *Addison Wesley*, 1994.

[16] Joost blog. http://blog.joost.com/2008/10/and_were_off_1.html.

[17] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in P2P systems. In *Proc. of P2P*, 2003.

[18] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proc. of SOSP*, pages 298–313, New York, NY, USA, 2003. ACM.

[19] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proc. of IMC '07*, New York, NY, USA, 2007.

[20] J. Chakareski, S. Han, and B. Girod. Layered coding vs. multiple descriptions for video streaming over multiple paths. In *Multimedia Systems, Springer, online journal*, 2005.

[21] I. Chatzidrossos, G. Dán, and V. Fodor. Delay and playout probability trade-off in mesh-based peer-to-peer streaming with delayed buffer map updates. In *Peer-to-peer Networking and Applications*, 2009.

[22] S. Cheshire. Latency and the quest for interactivity. *White paper commissioned by Volpe Welty AssetManagement, L.L.C., for the Synchronous Person-to-Person Interactive Computing Environments Meeting*, 1996.

[23] Y. R. Choe, D. Schuff, M. D. Jagadeesh, and V. S. Pai. Improving VoD server efficiency with BitTorrent. In *Proc. of MULTIMEDIA*, pages 117–126, New York, NY, USA, 2007. ACM.

[24] Y. Chu, J. Chuang, and H. Zhang. A case for taxation in peer-to-peer streaming broadcast. In *Proc. of PINS*, 2004.

[25] Y. Chu, A. Ganjam, T. S. E. Ng, S. Rao, K. Sripanidkulchai, J. Zhan, , and H. Zhang. Early experience with an internet broadcast system based on overlay multicast. In *Proc. of USENIX*, 2004.

[26] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2000.

[27] Y.-H. Chu and H. Zhang. Considering altruism in peer-to-peer internet streaming broadcast. In *Proc. of IEEE NOSSDAV*, 2004.

[28] D. Ciullo, M. A. Garcia, A. Horvath, E. Leonardi, M. Mellia, and D. Rossi. Network awareness of p2p live streaming applications. In *Proc. of HotP2P*, 2009.

[29] D. Ciullo, M. Mellia, M. Meo, and E. Leonardi. Understanding p2p-tv systems through real measurements. In *Proc. of Globecom*, 2008.

[30] B. Cohen. Incentives build robustness in bittorrent. In *Proc. of P2P ECON*, 2003.

[31] D. Croce, M. Mellia, and E. Leonardi. The quest for bandwidth estimation techniques for large-scale distributed systems. In *Proc. of Hotmetrics*, 2009.

[32] A. da Silva, E. Leonardi, M. Mellia, and M. Meo. A bandwidth-aware scheduling strategy for p2p-tv systems. In *Proc. of International Conference on Peer-to-Peer Computing*, pages 279–288, Washington, DC, USA, 2008. IEEE Computer Society.

[33] DailyMotion. `http://www.dailymotion.com`.

[34] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over peers. Technical report, Standford University, 2001.

[35] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the ip multicast service and architecture. In *IEEE Network magazine special issue on Multicasting*, 2000.

[36] Cisco Visual Networking Index: Forecast and 2008-2013 Methodology. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html`.

[37] Gnutella Forum. `http://groups.yahoo.com/group/the_gdf/`.

[38] D. Francey. Prototypage de système de vod décentralisé, 2009.

[39] P. Francis. Yoid: Extending the internet multicast architecture. In *Technical report, ACIRI*, 2000.

[40] A. Gai, F. Mathieu, F. de Montgolfier, and J. Reynier. Stratification in P2P networks: Application to bittorrent. In *Proc. of ICDCS*, 2007.

[41] A. Gai and L. Viennot. Incentive, resilience and load balancing in multicasting through clustered de bruijn overlay network (prefixstream). In *Proc. of IEEE International Conference on Networks (ICON)*, volume 2, pages 1–6. IEEE Computer Society, September 2006.

[42] A.-T. Gai and L. Viennot. Prefixstream: A balanced, resilient and incentive peer-to-peer multicast algorithm. Technical report, INRIA research repor 5514, 2005.

[43] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), 2003.

[44] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *Proc. of IEEE INFOCOM*, 2003.

[45] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *Proc. of IMC*, 2005.

[46] L. Guo, E. Tan, S. Chen, Z. Xiao, and X Zhang. The stretched exponential distribution of internet media access patterns. In *Proc. of PODC*, 2008.

[47] Y. Guo, C. Liang, and Y. Liu. Adaptive queue-based chunk scheduling for p2p live streaming. In *Proc. of IFIP Networking*, 2008.

[48] Y. J. Hall, P. Piemonte, and M. Weyant. Joost: A measurement study. Technical report, School of Computer Science, Carnegie-Mellon University, may 2007.

[49] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. A measurement study of a large-scale P2P IPTV system. In *IEEE Transactions on Multimedia*, 2007.

[50] X. Hei, Y. Liu, and K.W. Ross. Inferring network-wide quality in p2p live streaming systems. In *IEEE JSAC*, 2007.

[51] C. Huang, J. Li, and K.W. Ross. Can internet video-on-demand be profitable? In *Proc. of ACM SIGCOMM*, 2007.

[52] Y. Huang, Z.J. Fu, D.M. Chiu, J.C.S. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p vod system. In *Proc. of ACM Sigcomm*, 2008.

[53] UUsee Inc. `http://www.uusee.com/`.

[54] V. Janardhan and H. Schulzrinne. Peer assisted VoD for set-top box based IP network. In *Proc. of Peer-to-Peer Streaming and IP-TV Workshop (P2P-TV)*, pages 1–5, 2007.

[55] Joost. `http://www.joost.com/`.

[56] A.-M. Kermarrec, E. Le Merrer, Y. Liu, and G. Simon. Surfing peer-to-peer iptv system: Distributed channel switching. In *Proc. of EuroPar*, 2009.

[57] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. of ACM SOSP*, 2003.

[58] N. Laoutaris, D. Carra, and P. Michiardi. Uplink allocation beyond choke/unchoke or how to divide and conquer best. In *Proc. of CoNEXT 2008, 4th ACM International Conference on emerging Networking Experiments and Technologies, December 9, 2008, Madrid, Spain*, Dec 2008.

[59] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in bittorrent systems. In *Proc. of ACM SIGMETRICS*, 2007.

[60] A. Legout, G. Urvoy-Keller, and P. Michiardi. Understanding bit- torrent: An experimental perspective. Technical Report 00000156, INRIA, 2005.

[61] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the new coolstreaming: Principles, measurements and performance implications. In *Proc. of IEEE INFOCOM*, 2008.

[62] C. Liang and Y. Guo, Y.and Liu. Is random scheduling sufficient in p2p video streaming? In *Proc. of ICDCS*, 2008.

[63] W.S. Lin, H.V Zhao, and K.J.R. Liu. A game theoretic framework for incentive-based peer-to-peer live-streaming social networks. In *Proc. of ICASSP*, 2008.

[64] S Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang. Performance bounds of peer-assisted live streaming. In *Proc. of ACM SIGMETRICS*, 2008.

[65] Y. Liu. On the minimum delay peer-to-peer video streaming: how realtime can it be? In *Proc. of International conference on multimedia*, 2007.

[66] Z. Liu, Y. Shen, S. Panwar, K.W. Ross, and Y. Wang. Using layered video to provide incentives in p2p streaming. In *Proc. of Sigcomm P2P-TV Workshop*, 2007.

[67] R.T.B. Ma, S.C.M. Lee, J.C.S. Lui, and D.K.Y. Yau. Incentive and service differentiation in p2p networks: A game theoretic approach. In *IEEE/ACM Transactions on Networking*, 2006.

[68] N. Magharei and R. Rejaie. Prime: peer-to-peer receiver-driven mesh-based streaming. In *Proc. of IEEE INFOCOM*, 2007.

[69] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live p2p streaming approache. In *Proc. of IEEE INFOCOM*, 2007.

[70] P. Marciniak, N. Liogkas, A. Legout, and E. Kohler. Small is not always beautiful. In *Proc. of the Seventh International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[71] L. Massoulié. Peer-to-peer live streaming: Optimality results and open problems. In *Proc. of IEEE CISS*, 2008.

[72] L. Massoulié, A. Twigg, C. Gkantsidis, and P. Rodriguez. Decentralized broadcasting algorithms. In *Proc. of IEEE INFOCOM*, 2007.

[73] L. Massoulié, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized decentralized broadcasting algorithms. In *Proc. of IEEE INFOCOM*, 2007.

[74] L. Massoulié and M. Vojnović. Coupon replication systems. *SIGMETRICS Perform. Eval. Rev.*, 33(1):2–13, 2005.

[75] F. Mathieu. Heterogeneity in distributed live streaming: Blessing or curse? Technical report, Orange Labs Research Report RR-OL-2009-09-001, 2009.

[76] F. Mathieu and J. Reynier. Missing piece issue and upload strategies in flashcrowds and P2P-assisted filesharing. In *Proc. of P2PSA*, 2006.

[77] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. of IPTPS*, pages 53–65, London, UK, 2002. Springer-Verlag.

[78] P. Maymounkov and D. Mazieres. Rateless codes and big downloads, 2003.

[79] NapaWine. `http://www.napa-wine.eu/`.

[80] T. Nguyen and A. Zakhor. Distributed video streaming with forward error correction. In *Proc. of Packet Video Workshop*, 2002.

[81] J. Noh, P. Baccichet, and B. Girod. Experiences with a large-scale deployment of the stanford peer-to-peer multicast. In *Proc. of International Packet Video Workshop*, 2009.

[82] J. Noh, P. Baccichet, A. Hartung, F. Mavlankar, and B. Girod. Stanford peer-to-peer multicast (sppm) – overview and recent extensions,. In *Proc. of International Picture Coding Symposium (PCS)*, 2009.

[83] P2P-Next. `www.p2p-next.org/`.

[84] P2PIm@ges. `http://p2pimages.devoteam.com/`.

[85] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulcha. Distributing streaming media content using cooperative networking. In *Proc. of International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2002.

[86] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proc. of 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[87] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson. Analysis of bittorrent-like protocols for on-demand stored media streaming. In *Proc. of ACM SIGMETRICS*, pages 301–312, 2008.

[88] Peerialism. `http://www.peerialism.se/`.

[89] F. Pianese. *PULSE - an adaptive practical live streaming system*. PhD thesis, Eurecom, 2007.

[90] F. Picconi and L. Massoulié. Is there a future for mesh-based live video streaming? In *Proc. of the Eighth International Conference on Peer-to-Peer Computing*, pages 289–298, Washington, DC, USA, 2008. IEEE Computer Society.

[91] F. Picconi and L. Massoulie. Isp-friend or foe? making p2p live streaming isp-aware. In *Proc. of ICDCS*, 2009.

[92] PlanetLab. `http://www.planet-lab.org`.

[93] Grid5000 (G5K) Testbed Platform. `https://www.grid5000.org`.

[94] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In *Proc. of 4th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Feb 2005.

[95] PPLive. `http://www.pplive.com`.

[96] PPStream. `http://www.ppstream.com`.

[97] Meridian Project. `http://www.cs.cornell.edu/People/egs/meridian/`.

[98] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proc. of ACM SIGCOMM*, pages 367–378, New York, NY, USA, 2004. ACM.

[99] L Rizzo. Effective erasure codes for reliable computer communication protocols. In *Computer Communication Review*, 1997.

[100] S. Sanghavi, B. Hajek, and L Massoulié. Gossiping with multiple messages. In *Proc. of IEEE INFOCOM*, 2007.

[101] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking*, 2002.

[102] M. Schiely and P. Felber. Crossflux: an architecture for peer-to-peer media streaming. In *Global Data Management*, 2006.

[103] Orange VoD service. `http://abonnez-vous.orange.fr/residentiel/tv/video-demande.aspx`.

[104] T. Silverston and O. Fourmaux. Measuring p2p iptv systems. In *Proc. of NOSSDAV*, 2007.

[105] SopCast. `http://www.sopcast.com/`.

[106] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M. F. Kaashoek, f. Dabek, and h. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[107] K. Suh, C. Diot, J. Kurose, L. Massoulié, C. Neumann, D. Towsley, and M. Varvello. Push-to-Peer Video-on-Demand system: design and evaluation. *IEEE JSAC*, 25(9), 2007.

[108] Y.-W. Sung, M. Bishop, and S. G. Rao. Enabling contribution awareness in an overlay broadcasting system. In *Proc. of ACM SIGCOMM*, 2006.

[109] Akamai Technologies. `http://www.akamai.com`.

[110] S. Tewari and L. Kleinrock. Analytical model for bittorrent-based live video streaming. In *Proc. of Consumer Communications and Networking Conference (CCNC)*, 2007.

[111] Y. Tian, D. Wu, and K.W. Ng. Modeling, analysis and improvement for bittorrent-like file sharing networks. In *Proc. of IEEE INFOCOM*, 2006.

[112] Van Jacobson Google Talk "A New Way to look at Networking". `http://video.google.com/videoplay?docid=-6972678839686672840#`.

[113] D.A. Tran, K. A. Hua, and T. T. Do. A peer-to-peer architecture for media streaming. In *IEEE JSAC*, volume 22, 2004.

[114] TVants. `http://tvants.en.softonic.com/`.

[115] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured end system multicast. In *Proc.of ICNP*, 2006.

[116] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt. Measurement of a large-scale overlay for multimedia streaming. In *Proc. of HPDC*, 2007.

[117] F. Wang, J. Liu, and Y. Xiong. Stable peers: Existence, importance, and application in peer-to-peer live video streaming. In *Proc. of IEEE INFOCOM*, 2008.

[118] M. Wang and B. Li. Network coding in live peer-to-peer streaming. In *IEEE Transactions on Multimedia*, 2007.

[119] M. Wang and B. Li. R2: random push with random network coding in live peer-to-peer streaming. In *IEEE JSAC*, 2007.

[120] Application-Layer Traffic Optimization (ALTO) IETF working group. `http://www.ietf.org/html.charters/alto-charter.html`.

[121] C. Wu, B. Li, and S. Zhao. Multi-channel live p2p streaming: Refocusing on servers. In *Proc. of IEEE INFOCOM*, 2008.

[122] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4p: provider portal for applications. In *Proc. of ACM SIGCOMM*, 2008.

[123] X. Yang and G. de Veciana. Service capacity of peer to peer networks. In *Proc. of IEEE INFOCOM*, 2004.

[124] YouTube. `http://www.youtube.com`.

[125] H. Yu, D. Zheng, B . Y. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. In *Proc. of EuroSys*, pages 333–344, New York, NY, USA, 2006.

[126] J. Yu, M. Li, F. Hong, and G. Xue. Free-riding analysis of bittorrent-like peer-to-peer networks. In *Proc. of IEEE Asia-Pacific Conference on Services Computing APSCC*, pages 534–538, Washington, DC, USA, 2006. IEEE Computer Society.

[127] M. Zhang, J.G. Luo, L. Zhao, and S.Q. Yang. A peer-to-peer network for live media streaming using a push-pull approach. In *Proc. of MULTIMEDIA*, 2005.

[128] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang. Optimizing the throughput of data-driven peer-to-peer streaming. In *Lecture Notes in Computer Science*, volume 4351, 2007.

[129] M. Zhang, Q. Zhang, L. Sun, and S. Yang. Understanding the power of pull-based streaming protocol: Can we do better? In *IEEE JSAC, special issue on Advances in Peer-to-Peer Streaming Systems*, 2007.

[130] X. Zhang, J. Liu, B. Li, and T. P. Yum. Coolstreaming/donet: A data-driven overlay network for live media streaming. In *Proc. of IEEE INFOCOM*, 2005.

[131] B. Q. Zhao, J. C.S. Lui, and D. Chiu. Mathematical modeling of incentive policies in p2p systems. In *Proc. of NetEcon*, 2008.

[132] Y. P. Zhou, D. M. Chiu, and J. C. S. Lui. A simple model for analysis and design of P2P streaming protocols. In *Proc. of IEEE ICNP*, October 2007.

List of Publications

[133] Y. Boufkhad, F. Mathieu, F. de Montgolfier, D. Perino, and L. Viennot. Fine tuning of a distributed video-on-demand system. In *Proc. of International Conference on Computer Communications and Networks (ICCCN)*, August 2009.

[134] N. Hegde, F. Mathieu, and D. Perino. Diffusion Épidémique de chunks en quasi-direct : la taille compte. In *Proc. of Rencontre francophone sur les aspects algorithmiques de télécommunications (ALGOTEL)*, June 2009.

[135] L. Muscariello, D. Perino, and D. Rossi. Do next generation networks need path diversity? In *Proc. of IEEE International Conference on Communications (ICC)*, June 2009.

[136] Y. Boufkhad, F. Mathieu, F. de Montgolfier, D. Perino, and L. Viennot. An upload bandwidth threshold for peer-to-peer video-on-demand scalability. In *Proc. of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2009.

[137] L. Muscariello, B. Nardelli, and D. Perino. Towards real implementations of dynamic robust routing paradigms exploiting path diversity. In *Proc. of IEEE International Conference on Ultra Modern Telecommunications (ICUMT)*, 2009.

[138] L. Muscariello and D. Perino. Evaluating the performance of multi-path routing and congestion control in presence of network resource management. In *Proc. of IEEE International Conference on Ultra Modern Telecommunications (ICUMT)*, 2009.

[139] L. Muscariello and D. Perino. Modeling multi-path routing and congestion control under fifo and fair queuing. In *Proc. of IEEE Conference on Local Computer Networks (LCN)*, 2009.

[140] F. Benbadis, N. Hegde, F. Mathieu, and D. Perino. Playing with the bandwidth conservation law. In *Proc. of International Conference on Peer-to-Peer Computing (P2P)*, September 2008.

[141] T Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg. Epidemic live streaming: Optimal performance trade-offs. In *Proc. of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2008.

[142] Y. Boufkhad, F. Mathieu, F. de Montgolfier, D. Perino, and L. Viennot. Achievable catalog size in peer-to-peer video-on-demand systems. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2008.

[143] F. Pianese, D. Perino, J. Keller, and E. Biersack. PULSE: an adaptive, incentive-based, unstructured P2P live streaming system. *IEEE Transactions on Multimedia, Special Issue on Content Storage and Delivery in Peer-to-Peer Networks, Volume 9 N 6*, November 2007.

[144] F. Pianese and D. Perino. Resource and locality awareness in an incentive-based p2p live streaming system. In *Proc. of Sigcomm P2P-TV Workshop*, August 2007.

[145] D. Perino and F. Mathieu. Distribution trees' analysis of pulse, an unstructured p2p live streaming system. In *Proc. of Rencontre francophone sur les aspects algorithmiques de télécommunications (ALGOTEL)*, May 2007.

[146] D. Perino. The Pulse system - A new P2P prototype for live streaming, 2006.

This list of publications also contains the work we have carried on multi-path routing ([135, 138, 139, 137]), which has not been presented in this thesis.

Appendix

Appendix A

Synthèse en français

A.1 Introduction

Le *multimédia streaming* est devenu de plus en plus populaire ces dernières années. Le trafic média est doublé chaque 3-4 mois [2] et les prévisions indiquent qu'il doit augmenter de dix fois d'ici 2013 [36]. Contrairement au trafic Web, les données média peuvent être transférées par différentes architectures: IP multicast, multicast overlay, Content Distribution Networks (CDN), pair-à-pair (P2P), ... Ces solutions peuvent dépasser les limitations imposées par des architectures centralisées comme le client-serveur, où la bande passante et la capacité de stockage ne sont pas toujours suffisantes pour servir une audience à grande échelle.

L'IP multicast est théoriquement la meilleure solution pour une diffusion point-multipoint. Toutefois, il n'est pas largement déployé par les FAI à cause d'un manque d'intérêt commercial. Donc, l'IP multicast n'étant pas disponible à l'heure actuelle, il est nécessaire de choisir la solution qui est la plus adaptée aux caractéristiques du trafic media parmi les autres disponibles.

Une étude [46] effectuée en analysant plusieurs jeux de données montre que *la popularité des contenus média suivrait une distribution exponentielle étirée (stretch exponential)*, avec deux paramètres qui représentent l'âge des contenus et leur taille. Pour supporter des contenus qui suivent cette distribution les auteurs prouvent qu'un système doit avoir des capacités de caching qui augmentent en taille avec le nombre d'utilisateurs pour une longue période. Ils concluent en disant que **une architecture pair-à-pair streaming est efficace pour transférer des données média**. En effet, les ressources d'un système P2P augmentent avec le nombre d'utilisateurs, ce qui peut satisfaire les besoins du croissant du trafic média.

Prenant en compte l'évolution du trafic media et le potentiel que l'architecture pair-à-pair offre, nous étudions dans cette thèse des solutions P2P pour le streaming multimédia. Plus précisément, nous analysons l'approche pair-à-pair mesh pour le *live streaming*, et une solution complètement pair-à-pair (sans serveur central) pour la vidéo à-la-demande.

A.1.1 Multimédia et pair-à-pair

La façon la plus simple pour transférer un contenu multimédia est le **bulk media transfer** qui a été aussi la première solution utilisée. Avec cette approche les fichiers média sont considérés

comme des fichiers communs et peuvent être récupérés par FTP ou HTTP, ou encore téléchargés en utilisant des applications comme BitTorrent, E-donkey etc...

Avec le **streaming media**, le contenu est codé sous forme d'un flot continu de données qui est caractérisé par un certain *stream bitrate*: ce débit représente le débit de codage du flot et indique la quantité de données qui doit être récupérée chaque seconde pour décoder et jouer le stream[1].

Dans cette thèse nous allons étudier deux types réprésentatifs de streaming: le *on-demand streaming (streaming à la démande)* et le *live streaming (streaming en temps réel)*.

Dans le cas du **streaming à la demande**, un contenu est disponible à l'avance et il fait normalement partie d'un catalogue de contenus. Ce *catalogue doit être disponible, de sorte que chaque utilisateur puisse accéder à n'importe quel contenu n'importe quand*. Un système streaming à la demande doit *minimiser le temps que l'utilisateur doit attendre, une fois qu'il a choisi le contenu, avant que le stream commence à être joué (start-up delay)*. Une fois le stream démarré, le système doit assurer qu'il n'y aura pas de blocage pendant que la lecture : le débit de téléchargement du contenu doit par conséquence être compatible avec celui du stream.

Dans le cas du **streaming en temps réel**, un contenu n'est pas disponible à l'avance car il est généré pendant que l'événement à transmettre se déroule. Il y a donc une contrainte temporelle supplémentaire : le système doit *minimiser le temps qui passe entre la génération du contenu et le moment où le contenu est vu par les utilisateurs (play out delay)*. Étant donné que les contenus ne sont pas disponibles a l'avance, il n'y a pas vraiment de contrainte de stockage, mais les contraintes sur le *start-up delay et la continuité du stream* sont toujours importantes.

De cette analyse on peut comprendre qu'au delà de la qualité du flot, les contraintes le plus importantes dans le cadre du streaming media sont le **délai et la continuité du stream**. Si on est dans le cadre de l'on-demand streaming il y a la contrainte supplémentaire du **stockage du catalogue**. Les ressources qui posent des limites pour satisfaire ces besoins sont principalement la **bande-passante et la capacité de stockage du système**.

Les algorithmes qui s'occupent de gérer ces ressources sont donc des éléments clefs pour la performance du système.

Si une architecture pair-à-pair est utilisée, la bande passante disponible et la capacité de stockage augmentent avec le nombre d'utilisateur. Par contre, ces ressources sont distribuées et donc leur gestion est plus difficile que dans une architecture client-serveur. En plus, d'autres contraintes supplémentaires entrent en jeu.

Tout d'abord il faut construire et maintenir un réseau logique (overlay) : les nœuds sont volatiles et ils peuvent être nombreux, ce qui demande des algorithmes capable de garantir l'auto-organisation, la robustesse et la fiabilité de l'overlay. Des mécanismes pour l'échange et la représentation des données sont également nécessaires.

Les systèmes pair-à-pair peuvent être classés en deux catégories: *structuré* et *non-structuré*.

Dans un **réseau P2P structuré** la topologie et les relations entre les différents nœuds sont strictement définis. Ces systèmes utilisent normalement une table de hachage distribuée (DHT) avec différentes structures, et le routage est fait par clés.

[1] Si des techniques de codage spécifiques sont utilisées, il est possible de décoder le flot même si le débit de réception est inférieur au débit du stream, soit par des mécanismes de redondance (e.g. FEC), soit en variant la qualité du stream en fonction de la quantité de données reçue (e.g. layer coding).

Dans un **réseau P2P non-structuré** les liens entre les différents pairs sont établis d'une façon arbitraire. Ces réseaux utilisent des systèmes de routage basés sur des mécanismes d'inondation (flooding), épidémiques (epidemic) ou gossip pour les échanges des messages et des données. Les réseaux non-structurés sont les plus utilisés dans les systèmes réels car, même s'ils peuvent parfois échouer à localiser certains contenus rares, ils génèrent beaucoup moins d'overhead que les réseaux structurés pour les objets populaires, et ils sont plus robustes face au dynamisme du réseau et des pairs.

Les architectures pair-à-pair peuvent être utilisées pour réaliser des applications dans des environnements *contrôlés* ou *non-contrôlés*.

Dans un **environnement contrôlé** les pairs ont un comportement prévisible et il n'y a pas besoin de mécanismes d'incitation à la collaboration. Par exemple, l'ensemble des set-top boxes et gateways gérés par un même FAI forment un environnement contrôlé où des services fondés sur le pair-a-pair peuvent être mis en place.

Dans un **environnement non-contrôlé**, comme c'est le cas quand une application est utilisée sur les ordinateurs des usagers, les pairs ont un comportement peu prévisible et fournissent une quantité variable et incertaine de ressources.

A.2 Bornes sur les performances des systèmes pair-à-pair

Dans un système *pair-à-pair* ou *peer-assisted*, la somme des ressources fournies par les pairs est égale aux ressources qu'ils utilisent. Les algorithmes sont conçus pour utiliser au mieux les ressources disponibles, mais ils ne peuvent pas dépasser les limitations imposées par leur quantité totale.

Si l'on considère la bande passante, qui est une ressource critique pour les applications multimédia, la somme des données «uploadées» par les pairs est égale à la quantité qu'ils téléchargent : c'est la *loi de conservation de la bande passante*.

Nous utilisons la loi de conservation de la bande passante comme point de départ pour déduire des bornes théoriques sur les performances qu'une application pair-à-pair peut atteindre. Nos résultats peuvent être utilisés pour la conception et le paramétrage de systèmes réels.

Nous considèrons un système P2P générique, hybride, constitué par trois classes de nœuds :

- **Leecher** nœuds qui utilisent effectivement le service (les utilisateurs réels).
- **Seeders** nœuds qui ne sont pas en train d'utiliser le service, mais peuvent fournir des ressources au système.
- **Servers** Certains nœuds supplémentaires dédiés au service, installés par exemple par le fournisseur de service.

On suppose que chaque nœud a une capacité d'upload consacrée au service, et que la bande passante totale du réseau est la somme des capacités dédiées de tous les nœuds. Les leechers sont les seuls nœuds qui utilisent le service et donc les seuls qui ont besoin d'un débit descendant.

Dans notre modèle nous considérons seulement les échanges effectifs des données et ne prenons pas en compte le surcoût de contrôle (overhead). De plus, nous supposons que la bande passante

est parfaitement exploitée par les algorithmes du système et nous ne prenons pas directement en compte la composante temporelle.

La qualité de service (QoS) perçue par les utilisateurs dépend de l'application et, si l'on ne considère que la bande passante, est seulement liée à leur débit de téléchargement. Par exemple dans le cadre du partage de fichiers le débit peut être élastique, dans le cadre du streaming à la demande, il doit permettre tout le temps d'assurer la lecture du stream, ou encore dans le cadre du streaming en temps réel il doit être ajusté en permanence au débit du stream.

En utilisant la loi de conservation de la bande passante, nous déduisons des bornes de performance pour ce genre d'applications dans le cas d'une redistribution uniforme/non-uniforme de la bande passante du système, et avec une population fixe ou dynamique de nœuds. Si les ressources du système sont connues, nous pouvons calculer la *performance de téléchargement des leechers*. Au contraire, si l'on considère un débit cible nous pouvons calculer les *conditions de faisabilité* pour lesquelles ce débit peut être atteint.

A.2.1 Distribution uniforme de la bande passante

Nous commençons notre analyse en supposant que la bande passante disponible dans le système est équitablement partagée entre tous les leechers. C'est la plus simple allocation possible qui peut être obtenue si chaque uploader divise sa bande passante équitablement parmi tous les leechers. Toutefois, il est irréaliste de supposer qu'un pair puisse uploader à tous les leechers simultanément : il est plutôt plus réaliste de supposer que chaque uploader choisit quelques pairs à qui il envoie du contenu, en partageant équitablement sa bande passante entre eux. Il en résulte une bonne approximation d'une allocation uniforme si les choix ne sont pas biaisés.

Si la **population des nœuds est fixe**, le nombre de leechers et de seeders est connu, ainsi que leurs bandes passantes ; des statistiques précises peuvent alors être obtenues pour une période de temps dans lequel la population n'a pas évoluée.

Dans ce scenario le débit de téléchargement d'un leecher sera tout simplement la bande passante totale du système divisée par le nombre de leechers. Au contraire, si l'on se fixe un débit cible on peut distinguer deux situations. Si la capacité d'upload moyen des leechers et des seeders est suffisante pour fournir à tous les leechers un débit de téléchargement égal ou supérieur au débit cible, le système est dit *scalable*, car il peut gérer un nombre illimité de leechers sans besoin de serveurs supplémentaires. Si la capacité d'upload moyenne des leechers est égale à celle des seeders, la condition de passage à l'échelle devient simplement que le ratio entre la capacité d'upload des nœuds et le débit du stream doit être supérieur au taux d'activité du service (la proportion de leechers sur la population des leechers et des seeders).

Au contraire, si la capacité d'upload moyen de leechers et de seeders n'est pas suffisante, le système n'est pas scalable et de plus, des serveurs dédiés sont nécessaires. Dans ce cas, on a un effet de levier, c'est-à-dire que la capacité d'upload des serveurs est démultipliée en fonction de la capacité des seeders et des leechers.

Pour étudier des **populations dynamiques**, on peut supposer que les pairs entrent dans le système comme leechers selon un processus d'une certaine intensité. Un pair reste dans l'état de leecher pour une certaine période de temps, nommée *leeching time*, et ensuite il devient seeder, état dans lequel il va rester pour une autre période de temps, nommée *seeding time*, avant de quitter le système.

Dans ce scenario on doit exprimer le nombre de leechers et seeders en fonction du seeding time et du leeching time, et de l'intensité d'arrivée. Pour faire cela, le plus simple est de supposer que le système est dans un régime stationnaire, et que le nombre de pairs est suffisamment grand pour considérer un modèle fluide. On peut aussi supposer que les pairs restent *leechers* le temps nécessaire pour télécharger une certaine quantité de données. Avec cette condition on peut exprimer le leeching time en fonction de cette quantité de données

Le débit de téléchargement qu'un système peut fournir aux leechers, ainsi que les conditions de faisabilité, sont calculés comme dans le cas d'une population fixe, mais dans ce scenario les paramètres clefs sont le temps de seeding et l'intensité d'arrivée, qui déterminent la bande passante fournie par les seeders ainsi que le nombre de leechers. Si le système est *scalable* on dit qu'il est dans une condition d'*over-provisioning* ; autrement le système ne passe pas à l'échelle mais on retrouve un effet de levier par rapport à la bande passante fourni par les serveurs.

Des exemples d'applications de ces bornes théoriques à des systèmes réels sont décrits dans le Chapitre 2.3.3.

A.2.2 Distribution non-uniforme de la bande passante

Nous considérons également des systèmes où les pairs ont différents débits de téléchargement. Cela se produit lorsque le processus de sélection entre les pairs n'est pas aléatoire, mais prend en compte les ressources des voisins, ou est basé sur des mécanismes d'incitation. L'allocation non-uniforme peut également être une conséquence de facteurs non liés aux algorithmes d'allocation des ressources : différentes capacités de téléchargement maximales, nombre d'applications fonctionnant sur un même hôte, RTT entre pairs etc...

Comme cas d'étude, nous considérons une allocation non-uniforme obtenue par des algorithmes de type *tit-for-tat* [30, 41] [144]. Ces algorithmes amènent les leechers à partager leur bande passante de préférence avec ceux qui leur envoient le plus de données.

Un bonne modélisation du tit-for-tat peut être obtenue au moyen d'un paramètre γ [126, 40] qui indique la proportion de bande passante qu'un leecher partage en fonction de son historique de téléchargement, la partie restante étant allouée uniformément. Par conséquence le débit de téléchargement d'un pair donné est en partie déterminé par sa capacité d'upload en fonction de ce paramètre γ. Les capacités des seeders, des serveurs et la partie allouée uniformément par les leechers serons au contraire équitablement partagées entre tous les leechers indépendamment de leurs capacités d'upload.

Dans ce scenario si le système est dans une condition d'*over-provisioning* il est toujours *scalable* et tous les pairs peuvent atteindre leurs débit de téléchargement maximal.

Si le système *n'est pas scalable* on peut étudier des cas particuliers où l'on peut calculer le débit de téléchargement ainsi que les conditions de faisabilité, analytiquement ou par moyen d'une évaluation numérique.

Si par exemple on considère un système où il n'y a **que des leechers**, et si l'on connaît soit la distribution de bande passante, soit le paramètre γ, on peut obtenir des statistiques précises.

Pour une distribution de bande passante quelconque avec paramètre γ fixé on obtient:

- $\gamma = 0$ correspond à l'allocation uniforme de bande passante ; le débit moyen de téléchargement de leechers correspond à la capacité d'upload moyenne des nouveaux arrivants.

- Si $\gamma = 1$, chaque nœud télécharge à sa propre vitesse d'upload. Dans ce cas la vitesse moyenne d'upload est la moyenne harmonique, si elle existe, des capacités d'upload des nouveaux arrivants.

Si l'on prend en compte une distribution discrète constituée de deux classes de bande passantes, on peut analyser le cas où l'une des deux classes a une capacité d'upload nulle, qui correspond à la présence de *free-riders* dans un système. On peut ainsi calculer le nombre maximal de free-riders que l'application peut supporter (voir Théorème 2.2).

Pour calculer les solutions avec d'autres valeurs de γ et différentes distributions de bande passante, on peut résoudre numériquement les équations. On observe que le tit-for-tat diminue logiquement le temps de présence dans le système des pairs avec le plus de bande passante, car ils terminent leur téléchargement plus rapidement. D'un côté cela améliore leur performance, mais d'un autre côté cela cause une dégradation de la performance globale du système. Nous observons aussi que plus la distribution est dispersée, plus la performance globale se dégrade en augmentant le paramètre de tit-for-tat.

Un autre scénario intéressant à analyser est le cas où les pairs doivent respecter un certain **share-ratio**. Le share-ratio est le rapport entre la quantité de données uploadées et la quantité de données téléchargées par un pair pendant son séjour dans le système. Il peut être considéré comme un indicateur pour évaluer la contribution d'un pair au service par rapport aux ressources qu'il a utilisé. Cette valeur va influencer le temps de séjour d'un pair en fonction de sa bande passante et notamment le temps qu'il va passer en état de seeder.

D'après une évaluation numérique, on observe que le share-ratio améliore la performance du système en réduisant le temps de téléchargement des leechers si l'allocation de la bande passante est uniforme. Cela est dû à l'augmentation de la capacité d'upload totale des seeders, conséquence de l'augmentation du temps de seeding.

Si l'on rajoute le mécanisme tit-for-tat à un mécanisme de share-ratio, on remarque qu'il augmente le temps moyen de téléchargement. La valeur moyenne du temps de téléchargement est principalement affectée par les performances des pairs à faibles capacités d'upload qui obtiennent des temps de téléchargement plus long si $\gamma > 0$. D'autre part, les pairs les plus riches comptent sur γ pour obtenir une réduction de leur temps de téléchargement par rapport à l'allocation uniforme, et ils ne sont presque pas affectés par le share-ratio.

A.2.3 États transitoires et validation du modèle fluide

Le modèle analytique qu'on a utilisé pour notre analyse est basé sur un régime stationnaire considéré dans sa limite fluide. Les effets de l'état initial, où il y a généralement pas de seeders, sont alors considérés comme négligeables, tout comme les effets des arrivées et des départs discrets dans le régime stationnaire. Toutefois, une analyse plus fine de l'évolution du système vers l'état d'équilibre et de ces perturbations peut fournir des indications précieuses pour le dimensionnement d'un réseau pair-à-pair.

Dans le Chapitre 2.5 nous présentons une analyse de ces phénomènes, ainsi qu'une validation de notre modèle, au moyen d'un simulateur à événements discrets.

Nous observons qu'un système traverse différentes phases transitoires avant d'atteindre le régime stationnaire :

- Comme les leechers arrivent dans le système, il y a une phase initiale où la population des leechers augmentent à une vitesse égale à l'intensité d'arrivée, avec un débit de téléchargement constant par leecher, et il n'y a pas de seeders.
- Au cours d'une deuxième phase, le nombre de seeders augmente au fur et à mesure que les leechers complètent leurs téléchargements. Les seeders augmentent la bande passante disponible et diminuent le temps de téléchargement ; par conséquent les transitions leechers-seeders augmentent.
- Après un certain temps, les premiers seeders commencent à quitter le système alors que la bande passante des seeders réduit la population des leechers. Cette réduction du nombre de leechers finit par réduire à son tour la population de seeders. Cela réduit la bande passante globale disponible, provoquant une augmentation des temps de téléchargement et du nombre de leechers. Avec l'augmentation du temps de leeching, le temps total dans le système augmente...
- L'amplitude de ces cycles s'atténuant avec le temps, après quelques itérations, un régime stationnaire est finit par être atteint.

Ces phases sont présentes dans tous les différents scénarios considérés, avec une durée plus ou moins longue en fonction des paramètres du système.

Le processus d'arrivée n'est en général pas déterministe, et nous avons considéré le cas d'un processus d'arrivée de Poisson avec différentes intensités. Nous observons l'existence d'oscillations permanentes au sein du régime stationnaire. Lorsque l'intensité des arrivées est grande, la bande passante globale est large et suffisante pour que les fluctuations soient négligeables. Lorsque l'intensité de l'arrivée est faible, les arrivées peuvent trouver une bande passante insuffisante, ce qui augmente le temps de leeching temporairement et provoque des oscillations plus importantes. Nous validons ainsi notre hypothèse de modélisation fluide pour valeurs d'intensité d'arrivée grandes.

Nous observons aussi que plus la bande passante d'upload est dispersée, plus la bande passante globale disponible est variable. Cette variabilité cause des plus grandes fluctuations sur l'évolution du leeching time et du nombre de leechers. Puisque le modèle fluide ne considère que le débit moyen de transfert, il est plus précis pour des distributions d'upload moins dispersées. Nous confirmons aussi que, si l'on augmente l'intensité des arrivées, l'effet de la dispersion de la bande passante d'upload est moins importante et le modèle fluide est toujours valable.

A.3 Streaming en temps réel basé sur un réseau mesh

Nous considérons le streaming en temps réel pair-à-pair dans des *environnements non contrôlés*, où les pairs ont des comportements imprévisibles et partagent des quantités différentes de bande passante. Les techniques de distribution doivent donc être **robustes** à des scénarios dynamiques et imprévisibles, et doivent **inciter** les pairs à la coopération afin d'avoir assez de bande passante dans le système pour fournir le contenu à tous les nœuds.

L'approche *mesh non-structurée* semble être la plus appropriée pour répondre à ces exigences : elle a été employée pour le déploiement des applications streaming commerciales les plus

populaires, et a prouvé son efficacité sur une plus grande population d'utilisateurs par rapport à une approche structurée - arbres multiples.

Avec cette approche mesh le stream n'est plus transmis comme un flot continu de données mais il est divisé en une série de morceaux (chunks), qui sont injectés dans le système par une source et sont ensuite échangés entre les pairs afin de récupérer la séquence complète et de jouer le contenu. La diffusion du contenu est donc gérée par des algorithmes d'échange de chunks utilisés localement par chaque pair, qui peuvent être décrits par des fonctions de sélection de chunk et de pair. Chaque morceau suit une trajectoire potentiellement distincte de la source aux nœuds, menant à un arbre de distribution différent pour chaque chunk.

Les ressources disponibles dans le système, notamment la bande passante qui est la ressource critique pour le streaming en temps réel, sont donc exploitées pour échanger des chunks entre pairs afin de les distribuer aussi rapidement que possible tout en garantissant la continuité de diffusion et en minimisant le délai.

Les algorithmes d'échange de chunks peuvent être classés en *push* ou *pull*, selon que ce soit le pair qui veut envoyer le chunk (l'émetteur) ou le pair qui doit recevoir le chunk (le récepteur) qui fait la sélection. Cette classification est très claire en théorie mais en pratique chaque échange est plutôt une *négociation* entre l'émetteur et le récepteur et donc une solution *hybride push-pull*. Ces algorithmes peuvent être aussi classés en deux catégories selon que le pair ou le chunk est sélectionné en premier.

Un autre paramètre important à prendre en compte est le codage du stream. Des techniques de *source coding* peuvent être appliquées au stream, pour permettre à un pair de jouer le stream même s'il ne reçoit pas tous les chunks. Différents types de codage peuvent être utilisés, du simple *Forward Error Correction (FEC)* qui permet de récupérer les chunks perdus, jusqu'au *layer coding* où la qualité du stream dépend du débit de réception. Récemment des techniques de *network coding* ont été proposées pour le streaming en temps réel. Ces techniques permettent de réduire l'overhead de signalisation et donc de mieux utiliser la bande passante disponible.

Dans des scénarios où les pairs ont des capacités d'upload différentes, les algorithmes de diffusion des chunks doivent prendre en compte les ressources partagées par les nœuds. En outre, les systèmes doivent être robustes car les nœuds peuvent *tricher* et avoir une *comportement égoïste* pour améliorer leur performance. Les pairs devront également être encouragés à fournir la plus grande quantité de bande passante possible, en utilisant par exemple des mécanismes à la *tit-for-tat*.

Récemment, une attention croissante a été consacrée à la conception de mécanismes *network-aware*. Ces mécanismes influencent la construction de l'overlay (cf [91, 122]), ou agissent directement dans le processus de sélection des pairs pour la transmission des données, comme dans [144]. Certains d'entre eux sont simplement fondés sur des mesurés de latence ou de bande passante, tandis que d'autres plus complexes se basent sur des trackers externes pour récupérer des informations topologiques.

A.3.1 Métriques pour l'évaluation des performances

Les paramètres les plus importants pour évaluer la performance des algorithmes d'allocation pour des applications streaming en temps réel sont le *délai de diffusion (diffusion delay)* et le *taux de diffusion (diffusion rate)*.

Le *taux de diffusion* indique la probabilité asymptotique qu'un pair reçoive un chunk donné, ou, de manière équivalente, la proportion de pairs qui reçoit un certain chunk. Réciproquement, le *taux de perte (chunk miss ratio)* est la probabilité asymptotique de manquer un chunk, qui correspond de manière équivalente à la fraction de pairs qui ne reçoivent pas un chunk donné.

Le *délai de diffusion* indique le temps nécessaire à un chunk en moyenne pour arriver aux pairs après qu'il a été généré par la source.

Un autre paramètre important est l'overhead, qui peut être dû à la construction et la maintenance de l'overlay, ou aux messages nécessaires pour la négociation des échanges de chunks.

Les propriétés moyennes des arbres de diffusion des chunk sont aussi intéressantes à analyser. En fait, chaque chunk suit un parcours différent de la source aux pairs, mais certaines propriétés moyennes, comme la longueur ou la largeur des arbres, peuvent être importantes pour comprendre la performance d'un système mesh pour le streaming en temps réel.

A.4 Évaluation expérimentale de PULSE

Nous commençons notre analyse des algorithmes d'allocation de ressources dans les systèmes mesh pour le streaming en temps réel en analysant si des mécanismes d'incitation sont efficaces pour ce genre d'application. En particulier, nous analysons si une solution mesh couplée avec des mécanismes d'incitation à la tit-for-tat est en mesure de satisfaire les exigences du streaming en temps réel.

À cette fin, nous procédons à une évaluation expérimentale de PULSE, un système mesh pair-à-pair pour le streaming en temps réel que nous avons conçu et développé. Les algorithmes d'allocation de ressources PULSE sont basés sur une sélection des pairs à la *tit-for-tat* et utilisent *rarest first* comme politique de sélection chunk. Ces stratégies ont été adaptées au contexte du streaming en temps réel en tenant compte du délai moyen de réception des nœuds dans la sélection des pairs, et en réduisant le choix des chunks sur une fenêtre temporelle glissante de morceaux. PULSE est open source et est maintenant développé dans le cadre des projets Napa-Wine [79] et P2Pim@ges [84].

A.4.1 Description du système

Dans PULSE tous les nœuds sont identiques, sauf la source, qui diffère car c'est le premier nœud qui distribue le flot d'origine. Un protocole non-structuré, randomisé, de type gossip [43] est utilisé pour diffuser information sur l'existence des pairs et est responsable de garantir la connectivité entre les nœuds. Sur la base de cette connaissance les nœuds s'associe temporairement entre eux pour échanger des données en générant un mesh de connexions.

Les algorithmes d'allocation de ressources pour la gestion des connexions de données sont basés sur une combinaison de deux mécanismes d'incitation : *optimistic tit-for-tat* et *excess-based altruistic*. Intuitivement, le premier mécanisme devrait favoriser la coopération entre les nœuds qui ont plus de ressources, tandis que l'autre devrait faciliter la découverte des pairs et permettre aux nœuds les plus riches de contribuer plus efficacement au système.

Plus en détail, tous les nœuds sélectionnent deux groups de voisins auxquels ils vont envoyer des données : le groupe *MISSING* et le groupe *FORWARD*. Cette sélection est effectuée a intervalles réguliers de temps, nommés *epoch*.

Le groupe MISSING est rempli en choisissant les nœuds qui ont envoyé la plus grande quantité de données au nœud local dans la dernière epoch. Un certain nombre de pairs est aussi choisi aléatoirement parmi les voisins qui ont des donnés intéressantes pour le nœud local. Cette sélection aléatoire peut être influencée par des mesurés de latence.

Chaque nœud enregistre les échanges qui ont eu lieu dans le passé avec ces voisins à travers un *history score*. Pour chaque voisin cette valeur est incrémentée chaque fois qu'un pair envoie des données au nœud local alors qu'il n'est dans aucun groupe d'échange, et il est décrémenté chaque fois que le nœud local choisi ce voisin pour remplir le group FORWARD. Le group FORWARD est rempli avec les pairs qui n'ont pas de données intéressantes pour le nœud local mais qui ont les plus grands history score.

La sélection des chunks est faite par les récepteurs d'abord, qui envoient aux émetteurs une liste de requêtes. Les chunks demandés sont choisis selon un mécanisme *rarest first*. Les émetteurs envoient les chunks qu'ils ont le moins envoyé parmi ceux demandés.

La source a un seul groupe d'échange, le group MISSING, qui est changé à chaque epoch. Les pairs sont choisis au hasard parmi ceux qui ont un délai de réception plus petit qu'un certain seuil. La source envoie tout simplement les chunks qu'elle a envoyés le moins pour garantir une diffusion uniforme des premières copies des chunks.

A.4.2 Évaluation des performances

L'évaluation des performances de PULSE est faite au moyen d'un prototype que nous avons développé. Un premier jeu d'expériences a été effectué sur Grid5000 [93], une plate-forme de tests contrôlée.

Pour ces expériences nous avons artificiellement limité la capacité d'upload des pairs et les avons divisés en différentes classes de bande passante. Les détails sur les expériences ainsi que les paramètres et les résultats sont décrits dans le Chapitre 4.3.

Nous observons que le délai moyen de réception pour chaque classe est très stable dans le temps avec des fluctuations minimes. Le résultat le plus intéressant est la forte corrélation entre la bande passante disponible d'une classe et le délai moyen de ses membres : les pairs qui contribuent avec le plus de bande passante atteignent un délai plus petit. D'un autre côté, moins une classe contribue, plus élevé est son délai de reception. Les classes apparaissent *triée par ressources*, avec une différence significative entre le délai moyen de chaque classe : nous remarquons aussi que *la différence de décalage devient plus élevée lorsque la capacité disponible d'une classe est plus petite que le débit du stream.*

Un autre point intéressant c'est que le délai de toutes les classes converge très rapidement vers une valeur stable et que sa variance augmente au fur et à mesure que la capacité d'upload d'une classe diminue. Ceci suggère qu'avoir plus de capacité d'upload non seulement réduit le délai moyen, mais aussi le rend plus stable dans le temps.

Le taux de perte des chunks est lui aussi lié à la capacité d'upload d'une classe. Toutefois, tous les classes ont un taux de perte suffisamment bas pour être récupéré par le mécanisme de FEC implémenté dans PULSE.

Nous constatons que les seuls échanges MISSING donnent en moyenne plus ou moins le débit du stream. Même lorsque les pairs peuvent fournir beaucoup plus, la bande passante n'est pas utilisée pour les échanges MISSING, et elle serait probablement gaspillée s'il n'y avait pas des connections FORWARD. Cela indique que les connexions MISSING fournissent en moyenne un flot régulier de données vers et depuis des pairs qui sont intéressés par la même fenêtre temporelle de données.

Un deuxième résultat c'est que les connexions FORWARD sont très importantes pour répartir la capacité excédentaire des classes les plus riches vers les plus pauvres. De plus, la bande passante des classes les plus pauvres est rarement utilisée pour des échanges FORWARD.

Ces résultats confirment que les algorithmes de PULSE exploitent correctement la capacité disponible : *les échanges tit-for-tat sont importants pour garantir des échanges proportionnels réciproques, alors que les échanges altruistes permettent de distribuer les ressources non utilisées équitablement à l'ensemble du système*. Ceci montre bien que les algorithmes de diffusion doivent utiliser des mécanismes d'incitation mais aussi garder une partie d'échanges altruistes. Les deux mécanismes sont importants et doivent être utilisés car ils jouent un rôle critique pour la stabilité du système et sa performance.

Nous remarquons que la longueur des arbres de diffusion des chunks est *courte et assez stable dans le temps*. On peut aussi apprécier le fait que *les premières couches des arbres sont en moyenne très larges*, et que la proportion moyenne de nœuds qui se trouvent placés dans les dernières couches est très faible. Nous remarquons que *les nœuds avec le plus de ressource se trouvent dans les premières couches de diffusion*. Tout ces résultats indiquent que les algorithmes de diffusion de PULSE sont efficaces pour placer les nœuds avec le plus de ressources près de la source, ce qui a comme effet de créer des premières couches de diffusion très larges et des arbres très courts.

Pour mieux comprendre les propriétés des arbres de diffusion nous analysons un scenario homogène où tous les pairs ont la même bande passante. Cela nous permet de comparer PULSE aux systèmes structurés, qui sont normalement proposés pour ce type de scenarios.

Nous observons que dans ce scenario *les premières couches des arbres de diffusion se comportent en moyenne comme un arbre binaire tandis que, en descendant vers les feuilles, le nombre de pairs ayant un seul enfant augmente*. Si l'on observe la position des nœuds dans les arbres de chunks consécutifs, nous observons que PULSE se comporte comme *un arbre multiple de degré deux*. En plus, si l'on regarde la charge d'upload des pairs, nous observons que tous les nœuds fournissent au système un débit très proche du débit du stream. Ceci est une autre caractéristique importante de plusieurs solutions structurées, qui assurent que la charge est *équitablement répartie entre les nœuds*. Bien sûr dans PULSE on peut observer des comportements bizarres car les arbres ne sont pas construits de manière explicite mais sont les résultats d'échanges locaux de chunks. Toutefois, les propriétés moyennes indiquent que PULSE a un comportement relativement similaire à des solutions structurées.

Si l'on analyse PULSE en présence *d'arrivées et départs massifs de pairs* on peut observer que *le système gère les deux cas sans impact sur les performances des taux de diffusions*. Bien sûr, le délai augmente ou diminue en fonction du nombre de pairs dans le système.

Dans le tableau 4.4 nous comparons la performance de délai de PULSE avec celles de certains systèmes structurés pour le streaming en temps réel, ainsi qu'avec la valeur du délai optimal qui peut être obtenu dans le scenario analysé. Nous observons que la performance de *PULSE est comparable à celles des systèmes structurés mais que le délai est deux fois la valeur optimale*.

PlanetLab

Nous avons déployé notre prototype sur PlanetLab où nous avons pu analyser ses performances dans un environnement réel. Sur cette plateforme nous n'avons pas limité la capacité d'upload des nœuds artificiellement, car elle est déjà limitée par des vrais goulots d'étranglement et surtout par la charge des CPUs des machines.

Les résultats, détaillés dans le Chapitre 4.4, montrent que PULSE se comporte raisonnablement bien, même dans cet environnement difficile, prouvant un haut niveau de robustesse et d'adaptabilité. On peut remarquer que 90 % des pairs ont un délai de réception de moins de 25 secondes, et que 50 % ont un délai inférieur à 10 secondes.

Sur PlanetLab nous avons aussi pu analyser l'efficacité d'un mécanisme qui influence la sélection des pairs en fonction de leurs latences réseau. Nous observons que ce mécanisme très simple est capable de localiser fortement le trafic et d'améliorer la performance globale du système.

A.5 Streaming épidémique en temps réel

Pour mieux comprendre le processus de diffusion dans un réseau mesh pour le streaming en temps réel nous considérons maintenant les algorithmes de diffusion d'un point de vu plus théorique. Nous nous focalisons sur les éléments clefs pour la diffusion, en analysant seulement les algorithmes d'échange de chunks sans considérer dans un premier temps les autres aspects. Le but de notre analyse est de comprendre si une diffusion optimale est possible dans un réseau mesh, et d'analyser les compromis de performances les plus importants dans ce type de réseaux.

A.5.1 Résultats d'optimalité

Il existe un compromis naturel entre le taux et le délai de diffusion. Le taux de diffusion est généralement maximisé par une diffusion homogène des chunks parmi les pairs, indépendamment de l'âge des chunks. Toutefois, une telle diffusion peut générer des délais de diffusion élevés. D'autre part, pour minimiser les délais de diffusion, la priorité devrait être donnée à la transmission des chunks les plus récents. Le prix à payer est un taux de diffusion potentiellement non-optimal, parce que des chunks ne sont plus retransmis par un pair une fois qu'il a reçu des chunk plus frais.

Zhang et al. [129, 128] étudient l'optimisation du throughput et du taux de diffusion, et il proposent des techniques distribuées d'optimisation ainsi que des algorithmes pull qui permettent un taux de diffusion quasi-optimal.

Massoulié et. al. [73] prouvent qu'une diffusion *most deprived peer / random useful chunk* peut obtenir un taux de diffusion optimal ; Sanghavi, Hajek et Massoulié [100] montrent que le *random peer/latest blind chunk* peut diffuser les chunks dans un délai optimal.

Le premier algorithme qui est prouvé comme étant optimal en taux et délai de diffusion est le *random peer / latest useful chunk*, qui est proposé dans [141]. Cet algorithme peut diffuser les chunks à tous les pairs dans un délai optimal, à une constante probabiliste près. Plus récemment, Abeni, Kiraly and Lo Cigno [3] ont proposé un algorithme qui peut diffuser un chunk à tous les pairs dans un délai exactement optimal.

A.5.2 Algorithmes pour systèmes à bande passante homogène

Nous commençons notre analyse par des systèmes homogènes, où tous les pairs ont la même capacité d'upload. Dans ces scenarios il n'est pas nécessaire de prendre en compte les ressources que les pairs fournissent au système car tous fournissent la même quantité de bande passante (les effets de latence sont négligés).

Nous supposons que les pairs peuvent envoyer un chunk par seconde et que c'est l'émetteur qui commence la négociation pour l'échange (push). Selon la capacité d'upload de la source, nous analysons trois régimes possibles :

- *underload regime*: la source émet et envoie moins d'un nouveau chunk par seconde ;
- *critical regime*: la source émet et envoie un nouveau chunk par seconde ;
- *overload regime*: la source émet et envoie plus d'un nouveau chunk par seconde.

Clairement dans un régime overload, les pairs ne peuvent recevoir qu'une fraction des chunks émis par la source.

Nous considérons les politiques suivantes de sélection pair/chunk :

Random peer : Le destinataire est choisi au hasard parmi les voisins ;

Random useful peer : Le destinataire est choisi au hasard parmi les voisins pour lesquels l'émetteur a des chunks utiles (chunks qui ne sont pas encore possédés par les voisins) ; si le chunk est déjà sélectionné, le choix se fait parmi les voisins qui ne possèdent pas ce chunk (s'il y en a) ;

Most deprived peer : Le destinataire est choisi au hasard parmi les voisins pour lesquels il y a le plus grand nombre de chunk utiles ;

Latest blind chunk : L'expéditeur choisit le chunk le plus récent qu'il a reçu ;

Latest useful chunk : L'expéditeur choisit le chunk le plus récent qu'il a reçu et que ne possède pas le destinataire. Si le chunk est choisi d'abord, c'est le chunk le plus récent pour lequel il y a des destinataires potentiel ;

Random useful chunk : L'expéditeur choisit un chunk au hasard parmi ceux pour lesquels il y a des destinataires potentiels.

Les algorithmes que nous analysons se déduisent tous des combinaisons de ces politiques de choix et sont listés dans le tableau 5.1. Pour l'analyse nous supposons par défaut que les pairs peuvent avoir une connaissance complète de l'overlay et des chunks reçus de tous leurs voisins.

A.5.3 Formules récursives

Nous proposons des formules récursives pour décrire le processus de diffusion obtenu par les algorithmes *latest blind chunk / random peer* et *latest blind chunk / random useful peer*. En détails, nous estimons l'évolution de la proportion de pairs qui ont reçu un chunk donné en fonction du temps.

Nous supposons que les pairs sont indépendants entre eux, ce qui peut être le cas s'ils sont en assez grand nombre. En plus, nous supposons que la probabilité qu'un chunk soit reçu par un pair est indépendante de la probabilité que les autres chunks soient reçu par le même pair. Nous supposons d'abord que le système est dans un régime *underloaded* ou *critical*.

Comme condition initiale nous supposons que la source envoie une copie d'un chunk fixé au hasard entre parmi pairs du système. Dans la suite nous allons expliquer rapidement comment nous avons trouvé ces formules récursives qui sont détaillées dans le chapitre 5.2.2. Une comparaison avec des résultats de simulations nous permet de conclure que nos formules sont très précises dans l'estimation du taux et du délai de diffusion.

Latest blind chunk / random peer

Les pairs qui ont reçu le chunk vont l'envoyer s'il est le plus récent parmi ceux qu'ils ont reçu. La probabilité qu'un chunk soit le plus récent peut être obtenue en multipliant la probabilité qu'un pair ait reçu le chunk considéré par la probabilité qu'il n'ait pas reçu de chunks plus récents.

Les pairs qui envoient le chunk considéré, choisissent un destinataire au hasard. Il faut donc tenir en compte que tous les transferts ne seront pas utiles : une partie des pairs qui vont recevoir le chunk l'auront déjà reçu, et un même pair peut recevoir simultanément plus d'une copie d'un même chunk.

Latest blind chunk / random useful peer

Avec cet algorithme les pairs qui ont déjà reçu le chunk vont l'envoyer si c'est le plus récent qu'ils ont reçu exactement comme dans le cas considéré avant. Par contre le mécanisme de sélection des destinataires assure que tous les transferts seront utiles.

Information retardée pour Latest blind chunk / random useful peer

Des messages de contrôle sont nécessaires pour mettre à jour les informations concernant les chunks que les voisins ont téléchargés. Ces messages peuvent ne pas être transmis instantanément pour réduire l'overhead. Le retard provoqué peut changer les performances des algorithmes. Nous modélisons ce retard, en supposant que les pairs ne sont pas au courant des échanges en cours.

Il faut donc prendre en compte le fait que plusieurs pairs peuvent envoyer un même chunk au même destinataire en même temps et donc des *collisions* sont possibles.

Overload regime

Dans ce régime, la source émet plus d'un chunk par second. Il faut donc prendre en compte l'interférence des multiples chunks émis pendant un même créneau. Il faut estimer leurs taux de diffusion respectifs et modifier les formules en fonction de ces taux.

A.5.4 Analyse de performance par simulations

Si on considère le régime *critical*, trois algorithmes ont des performances meilleures que les autres : *most deprived peer / latest useful chunk (md/lu)*, *latest useful chunk / most deprived peer (lu/md)* et *latest useful chunk / random useful peer (lu/up)*. Tous atteignent un taux de diffusion optimal, et tous ont un délai optimal ou quasi-optimal.

Le *random peer / latest useful chunk (rp/lu)* a un délai de diffusion assez élévé car il n'est pas optimal en régime critique.

On observe que la politique de sélection du dernier chunk reçu atteint des délais de diffusion optimaux, et, si elle est couplée avec une sélection utile, elle peut aussi atteindre un taux de diffusion optimal. Sélectionner le pair d'abord réduit le taux de diffusion parce que, une fois cette sélection effectuée, l'expéditeur n'est pas sûr d'avoir des chunks utiles pour le pair sélectionné. Cela peut être surmonté en sélectionnant un pair pour lequel ils existent des chunks utiles. Toutefois, une telle sélection utile, ou la sélection d'un chunk utile d'abord, nécessitent un overhead plus important car il est nécessaire de connaître avec précision l'état de tous les voisins par rapport à une sélection aveugle.

Si l'on varie *la taille du système (le nombre des pairs)* le taux de diffusion des algorithmes ne varie pas sauf pour *lb/up*. En fait ce dernier augmente son taux de diffusion avec le nombre de pairs et semble être asymptotiquement optimal.

Comme prévu le délai de diffusion augmente proportionnellement avec le nombre de pairs. Le délai que *dp/ru* atteint semble le plus affecté par l'augmentation de la taille du système.

Si l'on varie *la capacité d'upload de la source* on observe que *rp/lu* a des performances dégradées pour tous les valeurs proches du régime critique, alors que *dp/ru* et *dp/lu* ont des mauvaises performances en régime overload.

Une façon de réduire l'overhead est de réduire la taille du voisinage. Nous analysons trois techniques pour restreindre le nombre de voisins :

- *graph statique* avec 10 voisins par pair ;
- *graph aléatoire* avec pré-sélection aléatoire de deux pairs pour chaque envoi de chunk ;
- *graph adaptatif* où le dernier pair choisi est maintenu et un autre pair est choisi aléatoirement.

Le graph statique réduit fortement le performance en termes de taux de diffusion, alors que le graph adaptatif augmente le délai de diffusion. Le meilleur compromis est obtenu par le graph aléatoire pur.

A.5.5 Algorithmes resource-aware pour les systèmes hétérogènes

Le fonctionnement des algorithmes de diffusion de chunks en milieu hétérogène, où les pairs ont une bande passante différentes entre eux, est beaucoup moins clair que dans le cas homogène. En plus, les algorithmes qu'on a considérés jusqu'à maintenant ne prennent pas en compte les ressources que les pairs fournissent au système.

Des résultats de simulations pour ces algorithmes dans des scenarios hétérogènes sont détaillés dans le chapitre 5.3.

Nous observons que les performances empirent au fur et à mesure que l'hétérogénéité augmente. Si on regarde la distribution cumulée (CDF) des diffusions des différents chunks, on observe que dans un scenario homogène les valeurs sont concentrées autour de la moyenne, alors que dans un scenario hétérogène elles sont éparpillées sur une plage plus large. Nous observons que les performances de diffusion des chunks sont fortement conditionnées par les ressources des pairs qui reçoivent les premières copies des chunks. La dégradation des performances est liée au choix aléatoire des pairs, sans prise en compte des ressources investies dans le système. L'intuition est confirmée : les premières copies d'un chunk doivent être échangées entre les pairs avec le plus de ressources pour augmenter le taux de diffusion et diminuer le délai.

Nous considérons donc maintenant des algorithmes de diffusion qui prennent en compte les ressources partagées par les nœuds pour effectuer la sélection. Comme la ressource la plus importante dans les systèmes de streaming en temps réel est la bande passante, nous considérons des algorithmes qui essaient de prendre en compte la bande passante fournie par les pairs. Néanmoins, dans la section précédente nous avons souligné qu'un certain degré d'altruisme (sélection agnostique) est nécessaire pour le bon fonctionnement du système. Nous considérons ce compromis de sélection aware-agnostic et nous proposons un modèle qui prend en compte explicitement ce compromis, et qui peut représenter plusieurs algorithmes.

Nous nous concentrons sur le processus de sélection des pairs alors que pour la sélection des chunks nous considérons seulement *latest blind (lb)* and *latest useful (lu)*.

Nous proposons un modèle général qui permet de représenter différentes sélections de pairs non-uniformes. La sélection non-uniforme est représentée par des fonctions de poids. Chaque pair associe à chaque voisin un certain poids, qui représente d'une façon ou d'une autre les ressources que le voisin fourni au système. Chaque fois qu'un pair doit sélectionner un voisin il peut effectuer une sélection *aware* en fonction des poids, ou une sélection *agnostic* aléatoire parmi les voisins. La première est choisie avec une probabilité W nommée *awareness probability* qui indique à quel point un algorithme choisi les pairs en fonction des poids, et donc combien un algorithme est *resource-aware*. Le choix aléatoire est effectué avec la probabilité restante $1 - W$.

On va analyser trois politiques de sélection des pairs:

- **Random (rp)** le choix est toujours aléatoire, i.e. $W = 0$ (il n'y a pas besoin de définir de fonction de poids) ;
- **Bandwidth-aware (ba)** où les poids correspondent à la capacité d'upload des voisins.
- **Tit-for-tat (tft)** où les poids correspondent à la quantité de données fournies par chaque voisin à l'émetteur sur une période donnée de temps (*epoch*).

A.5.6 Formule récursives

Nous proposons des formules récursives pour décrire le processus de diffusion d'un algorithme générique *aware peer / latest blind chunk*. Comme nous l'avons vu, pour un milieu hétérogène, il est plus significatif de calculer les distributions des performances plutôt que leur valeur moyenne, et les premiers échanges des chunks, en particulier, sont déterminants pour la performance de la diffusion ; nous proposons donc une approche en deux étapes.

La première étape consiste à calculer de manière déterministe les évolutions possibles des premiers échanges d'un chunk jusqu'un certain temps. Dans une deuxième étape il est possible d'utiliser ces valeurs comme conditions initiales pour approximer la suite du processus de diffusion. Le nombre d'échanges à calculer doit être le plus petit possible pour minimiser le coût de calcul déterministe. En pratique, 4-5 échanges semblent suffisants pour avoir une bonne estimation des performances finales et donc l'approche récursive est plus intéressante qu'une analyse complète par simulation en termes de temps et de ressources de calcul.

L'approximation récursive est similaire au cas homogène avec la différence que dans le cas hétérogène les pairs sont choisis en fonction des poids. Les formules, qui doivent être appliquées sur toutes les condition initiales pour avoir une distribution des performances, sont décrites au chapitré 5.3.2. La comparaison avec des résultats de simulation montre que les formules sont relativement précise dans l'estimation de la performance; toutefois, les erreurs d'estimation sont plus grandes que dans le cas homogène.

A.5.7 Analyse de performance par simulation

Nous présentons maintenant une évaluation des performances des algorithmes resource-aware au moyen d'un simulateur à événements. Le détail des simulations, ainsi que leurs paramètres, sont présentés dans le chapitre 5.3.3.

Nous observons qu'une certaine *awareness* des ressources (autour du 10%) est nécessaire pour améliorer les performances du système par rapport à une sélection aléatoire. Si l'on dépasse ce seuil nous observons un compromis délai/taux de diffusion en fonction du paramètre d'*awareness* W. Plus l'algorithme est *resource-aware* plus les performances des classes riches en bande passante sont meilleures. Par contre, les classes les plus pauvre reçoivent de moins en moins de chunks. Cela peut être considéré comme une bonne propriété du système car les pairs sont incités à contribuer plus pour améliorer leur performance. Par contre, il y a de la bande passante qui est gaspillée et qui pourrait être utilisée pour servir les classes pauvres. La bonne valeur W a choisir dépend donc de l'application et de son environnement de travail.

Un résultat intéressant est que la politique de sélection *tit-for-tat* a des performance très proches d'une sélection purement *bandwidth-aware*. Cette dernière est par construction la technique la plus précise pour prendre en compte les ressources que les nœuds fournissent au système. Par contre, il est très difficile de l'implémenter dans des systèmes réels car l'estimation de la valeur de bande passante est très difficile surtout quand beaucoup de nœuds effectuent les mesures en même temps. Au contraire, une politique *tit-for-tat* est très simple à implémenter et les performances sont seulement légèrement inférieures.

Dans le cadre d'une sélection *tit-for-tat* nous observons que, en augmentant la taille de l'epoch (la période de temps sur laquel la contribution des voisins est estimée), les performances glob-

ales du système s'améliorent. En fait, une plus longue epoch permet une meilleure estimation des contributions mais rallonge aussi la période de convergence du système.

Nous observons que la politique de sélection de la source a un fort impact sur la performance du système. Si la source est capable de connaître les ressources fournies par les pairs, et uploade les chunks systématiquement aux classes les plus riches, les performances sont les meilleures. Toutefois, l'estimation de la bande passante est difficile, et en particulier pour la source qui ne peut pas utiliser de tit-for-tat car elle ne télécharge pas de données.

Il est intéressant de remarquer que, si la source est un peu sur-dimensionnée en bande passante et qu'elle utilise une politique aléatoire de sélection des pairs, le système obtient des performances égales au cas où une source non sur-dimensionnée sélectionne toujours un pair avec beaucoup de bande passante. Par contre, dans le cas d'un choix aléatoire, il n'y a pas d'estimation de bande passante à faire et la sur-provision nécessaire est négligeable par rapport à la taille du système.

A.5.8 Optimisation des paramètres

Nous analysons maintenant l'impact que la taille des chunks et le nombre de pairs à sélectionner ont sur la performance du système. Pour cela, le simulateur a été modifié pour prendre en compte la latence entre les nœuds, et nous considérons des algorithmes où le pair est choisi d'abord.

Nous observons que les tailles possibles pour les chunks admettent un intervalle (*suitable range*) qui permet d'optimiser les performances pour les algorithmes considérés. Nos simulations montrent que choisir des chunks trop petits augmente trop l'overhead pour les échanges de contrôle et diminue le taux de réception. Au contraire utiliser des chunks trop grands augmente trop le délai sans améliorer la performance du système pour autant. À l'intérieur de cette zone utile intermédiaire, un compromis existe entre le taux et le délai de diffusion, et le choix final dépend des critères de QoS de l'application.

En modifiant les valeurs des latences entre les nœuds nous observons que cette zone utile dépend surtout de la valeur moyenne des RTTs et plus marginalement de l'algorithme exact considéré. Par contre, augmenter le nombre de connections d'upload en parallèle n'améliore pas vraiment le taux de réception mais augmente le délai.

Pour ce qui concerne le nombre de voisins nous observons que *contacter un nombre plus grand de voisins par rapport au nombre maximal de connections d'upload peut améliorer la performance* et élargir la zone utile.

A.6 Streaming à la demande

Les techniques et les algorithmes étudiés pour le streaming en temps réel ne peuvent pas être directement appliqués au streaming à la demande.

Dans le cadre du streaming à la demande :

- les contenus sont complètement disponibles avant l'utilisation du service. Ils peuvent donc être stockés, ce qui introduit les capacités de stockage du système comme paramètre supplémentaire.

- les utilisateurs ne sont pas synchronisés, de sorte que les contenus multimédia stockés devraient être toujours disponibles pour tous les clients. En outre, ce manque de synchronisation rend plus difficile l'échange de contenus entre les différents clients du service.

Les services les plus populaires de streaming à la demande sont la *Video-on-Demand (VoD)* et le *User Generated Content distribution (UGC)*. Le premier service offre habituellement aux utilisateurs un catalogue de fichiers multimédias (généralement des vidéos) : ce catalogue doit être aussi grand que possible afin d'attirer toutes sortes de clients, et les contenus doivent être toujours disponibles, de sorte qu'un abonné puisse accéder à chaque vidéo à tout moment. Ce service concerne par exemple les films ou les séries TV. Le deuxième type de service donne la possibilité aux clients de partager leurs propres contenus qui deviennent alors accessibles par tous les autres utilisateurs. UGC diffère de la VoD principalement par la fréquence à laquelle les contenus multimédia sont mis à jour, et par la distribution de popularité des contenus. En fait, dans un service UGC chaque jour des milliers de contenus sont uploadés (dont très peu sont regardés par d'autres utilisateurs) alors que dans un système de VoD le catalogue est mis à jour de temps en temps (une fois par semaine par exemple). Cela conduit à des techniques plus complexes pour le stockage et la maintenance du catalogue des contenus dans un service UGC.

Plusieurs architectures et algorithmes d'allocation des ressources ont été proposés pour satisfaire aux besoins et faire face aux contraintes de stockage et de bande passante du streaming à la demande. Une liste complète est disponible au chapitre 7. Toutes les solutions sont composées d'algorithmes d'allocation des contenus pour le stockage d'une part, et de gestion des connections pour la distribution des contenus d'autre part. Ces algorithmes peuvent être plus ou moins complexes selon le type d'architecture utilisée. De plus, les performances de toutes les architectures sont limitées par les capacité de stockage et de bande passante du système.

Nous considérons des systèmes de Video-à-la-Demande où il n'y a pas de serveur central pour la distribution proprement dite : les pairs participent au stockage du catalogue des vidéos et à la distribution des contenus pour satisfaire les requêtes des autres utilisateurs. En particulier, nous nous concentrons sur des algorithmes conçus pour des *environnements contrôlés*, comme un service de VoD déployé sur des set-top boxes installées au domicile des utilisateurs.

Nous avons vu que l'approche pair-à-pair peut augmenter le passage à l'échelle d'un service en terme de bande passante ; il peut aussi augmenter la capacité de stockage du système grâce au partage de la capacité de stockage de chaque pair. En plus, cette approche du streaming à-la-demande est économique car la bande passante et la capacité de stockage que chaque pair doit fournir sont assez faibles et disponibles aujourd'hui à des prix relativement bas. Ces ressources sont déjà disponibles par exemple dans les gateways et/ou set-top boxes que les FAI distribuent à leurs abonnés.

A.7 Taille du catalogue d'un système de vidéo-à-la demande

Nous analysons la taille du catalogue de vidéos qu'un système de vidéo à-la-demande peut fournir aux utilisateurs en prenant en compte les contraintes de capacité de stockage et de bande passante. Notre modèle étend celui présenté par Suh *et al.* [107], qui est basé seulement sur les contraintes de bande passante et qui ne considère pas la taille du catalogue.

A.7.1 Bande passante minimale

Nous analysons d'abord des systèmes qui ont une bande passante tout juste suffisante pour servir les requêtes des vidéos générées par les utilisateurs. Une borne triviale sur la taille du catalogue est donnée par la capacité de stockage de chaque utilisateur (exprimée en nombre de films) multipliée par le nombre d'utilisateurs, et est donc proportionnelle au nombre d'utilisateurs. Cette borne peut être atteinte si chaque vidéo est découpée en un nombre de parties égal au nombre d'utilisateurs (*full-striping*).

Toutefois, pour un système réel, il n'est pas possible de découper une vidéo en autant de morceaux, car par exemple, un client devrait se connecter à tous utilisateurs pour récupérer une vidéo donnée. Si l'on suppose qu'il y a un nombre maximal de connections qui peuvent être utilisées simultanément, et donc un nombre maximal de parties en lesquelles une vidéo peut être découpée, cette borne maximale ne peut pas être atteinte.

En fait, nous montrons dans le chapitre 8.1.2 que dans le cas où il y a un nombre limité de connections par pair, *la taille maximale possible est proportionnelle à la capacité de chaque utilisateur*. Nous proposons une allocation qui permet d'obtenir cette taille maximale de catalogue.

A.7.2 Bande passante légèrement sur-provisionnée

Si le système est légèrement sur-provisionné en bande passante alors même avec un nombre limité de connections *il est possible de stocker et fournir aux utilisateurs un catalogue dont la taille est proportionnelle a leur nombre*

Les théorèmes sont détaillés dans le chapitre 8.2. L'approche consiste à appliquer des arguments de *maximum flow* et des méthodes probabilistes pour montrer qu'une séquence de vidéos demandées peut être servie avec une probabilité élevée. Pour cela, il faut montrer que tous les graphes de *qui donne quoi* rencontrés dans la suite infinie des requêtes possibles ont une propriété expander. Ceci est possible grâce à la combinaison d'arguments algorithmiques concernant les restrictions sur la façon dont les demandes sont faites, et d'arguments probabilistes sur la façon dont les vidéos sont allouées.

Le résultat peut être adapté à des milieux hétérogènes où les pairs ont une bande passante et une capacité de stockage différentes entre eux. Dans ces scénarios des mécanismes de balancement sont nécessaires. Toutefois, ces résultats ne donnent pas directement des algorithmes pratiques distribués.

A.8 Algorithmes pratiques de vidéo à-la-demande

Nous proposons et analysons maintenant des techniques simples pour l'allocation et la distribution des vidéos dans un système pair-à-pair à-la-demande.

Nous supposons que les vidéos sont découpées en un certain nombre de stripes qui sont allouées et distribuées indépendamment. Nous supposons que chaque client a un cache où il stocke la vidéo qu'il est en train de télécharger et de regarder.

Comme techniques pour l'allocation des contenus nous considérons :

- *random (algorithme R)* toutes les vidéos sont répliquées le même nombre de fois et sont distribuées aléatoirement entre les clients.
- *popularity-based (algorithme P)* le nombre de répliquas d'une vidéo donnée dépend de sa popularité estimée.

Comme techniques pour la distribution des vidéos nous analysons :

- *Storage only (algorithme S)* les contenus ne peuvent être téléchargés qu'à partir des clients qui les stockent dans l'allocation originale. Un client accepte des requêtes de connection seulement s'il a encore de la bande passante disponible.
- *Caching and relaying (algorithme C)* les contenus peuvent être téléchargés soit à partir des clients qui les stockent dans l'allocation originale, soit à partir des clients qui les ont dans leurs caches. Un client accepte de nouvelles requêtes de connection seulement s'il a encore de la bande passante disponible.
- *Dynamic (algorithme D)* comme l'algorithme C sauf que les clients donnent priorité aux contenus du cache et peuvent supprimer une connexion existante pour un contenu de l'allocation originale s'ils reçoivent une requête pour un contenu dans le cache. Cet algorithme est étudié pour faire face à des contenus très populaires. Les pairs qui ont des connections supprimées doivent se reconnecter à d'autres clients pour continuer leur téléchargement.

A.8.1 Analyse de performance

Nous analysons les algorithmes listés dans le tableau 9.1 qui sont des combinaisons des techniques décrites au-dessus. Cette analyse est faite au moyen d'un simulateur à événements discrets et d'un prototype. Nous considérons des conditions extrêmes : un scenario de flash-crowd dans les heures pleines où tous les clients font une requête. La distribution de popularité et les fréquences d'arrivées des clients que nous utilisons proviennent de mesures réelles. Nous supposons que la popularité réelle des vidéos est un peu différente par rapport à l'estimation qui est utilisé pour l'allocation popularity-based.

Comme métriques de performance nous analysons la taille de catalogue maximal et le taux d'échec ; un client peut avoir un échec à cause d'un blocage pendant que la vidéo est jouée, ou peut tout simplement ne pas trouver assez de clients pour lui fournir le contenu. Les deux métriques (taille et taux d'échec) sont corrélées : si l'on se fixe un taux d'échec maximal, qui donc détermine la QoS du système, la taille du catalogue dépend du nombre de copies de chaque vidéo nécessaire pour satisfaire ce taux d'échec. À l'inverse, si l'on se fixe une taille de catalogue, on peut calculer le taux d'échec qu'il génère.

Nous observons d'abord que les algorithmes où *l'allocation est basée sur une estimation de la popularité des vidéos sont très sensibles à des erreurs d'estimation*, et déjà avec 10% d'erreur les performances sont fortement dégradées.

Au contraire, *l'allocation aléatoire couplée avec des mécanismes de caching permet d'obtenir de taille de catalogue assez importante*.

Nous vérifions également que *la bande passante disponible est une ressource critique* qui influence beaucoup la performance du système. En fait, en augmentant la bande passante, le taux d'échec diminue et la taille maximal du catalogue que le système peut supporter augmente. La bande passante peut être artificiellement augmentée en baissant le débit des contenus ; par contre la qualité des vidéos proposés sera alors impactée.

Si le taux d'échec permis est bas et si la bande passante disponible est faible, nous observons que des algorithmes dynamiques, qui donnent priorité aux vidéos stockées dans les caches des utilisateurs, améliorent la performance. Ce type d'algorithmes est aussi plus efficace si le nombre des clients est très grand ou s'il y a quelques vidéos extrêmement populaires.

Nous observons que *la plupart des autres paramètres jouent un rôle moins important* pour la performance. Il suffit en fait généralement de choisir des valeurs ni trop grandes ni trop petites pour garantir une bonne performance, par exemple pour ce qui concerne le nombre de stripes et le nombre de clients à contacter pour obtenir le contenu. Nous observons aussi que l'intensité des arrivées, si elle reste dans des valeurs raisonnables, n'impacte pas vraiment la performance du système, de même qu'une faible hétérogénéité.

A.9 Conclusion

L'architecture pair-à-pair constitue un potentiel énorme pour le déploiement de services et d'applications Internet. Les ressources d'un système pair-à-pair augmentent avec le nombre de participants, ce qui profite autant aux fournisseurs de services qu'aux utilisateurs. Des nouveaux services et applications peuvent être déployés à un coût très faible. L'inconvénient est que la nature décentralisé de cette architecture rend la gestion des ressources et des algorithmes plus compliquée à mettre en place.

La dernière décennie a vu le déploiement d'un large spectre d'applications pair-à-pair telles que le partage de fichiers, la téléphonie, le stockage, les jeux en ligne etc... Dans cette thèse nous avons considéré le multimedia streaming.

Nous avons d'abord quantifié le gain en terme de bande passante qu'une architecture pair-à-pair peut fournir par rapport à une approche centralisée. Nous avons mis en évidence que le fournisseur d'un service peut réduire ses coûts tout en augmentant considérablement sa capacité de service. Nous avons proposé des conditions de faisabilité pour des systèmes streaming en temps réel et à la demande, ainsi que pour des systèmes de partage de fichiers, et nous avons analysé le nombre de clients qu'une architecture pair-à-pair peut potentiellement servir.

Ensuite, nous nous sommes concentrés sur des systèmes pair-à-pair mesh pour le streaming en temps réel, conçu pour les environnements non contrôlés, et sur le streaming à la demande pour des environnements contrôlés.

Dans le cadre du streaming en temps réel nous avons montré qu'une approche mesh peut fournir une diffusion quasi-optimale et qu'il y a un compromis naturel entre le taux et le délai de diffusion. Nous avons aussi montré que cette approche couplée avec des mécanismes d'incitation est efficace pour le déploiement d'une application streaming en temps réel.

Nous avons montré que ces mécanismes d'incitation sont presque aussi efficaces que des technique bandwidth-aware tout en étant plus simples à mettre en place. Toutefois, un certain niveau

de diffusion uniforme aléatoire est nécessaire pour le bon fonctionnement de ces systèmes et le niveau d'awareness affecte le compromis entre taux et délai de diffusion.

Dans le cadre du streaming à-la-demande nous avons analysé la taille du catalogue de contenus qu'une approche pair-à-pair peut fournir aux utilisateurs. Nous avons montré qu'un catalogue de taille linéaire avec le nombre de clients est possible si le système est légèrement sur-provisionné en bande passante. Nous avons aussi analysé des algorithmes pratiques pour la mise en place d'un système pair-à-pair à-la-demande. Nous avons montré qu'une allocation aléatoire des contenus est efficace si elle est couplée avec des technique de caching, et que des algorithmes dynamiques peuvent augmenter la performance dans des scenarios critiques.

Perspectives

Le trafic streaming est en train d'augmenter fortement, et les prévisions indiquent qu'il devrait être multiplié par dix d'ici 2013. Dans cette thèse nous avons montré que les applications streaming peuvent augmenter le nombre d'utilisateurs à faible coût en utilisant des architectures basées sur le pair-à-pair. De plus en plus de fournisseurs pourront donc être intéressés par des architectures P2P pour faire face à de si grandes audiences.

Nous avons également montré que les architectures pair-à-pair sont très efficaces lorsqu'elles sont employées par les FAI dans des scénarios contrôlés, comme par exemple pour fournir des services à-la-demande hébergés sur les set-top boxes. Nous croyons que ces solutions sont très attractives pour les fournisseurs car le matériel nécessaire est déjà disponible. Au cours des prochaines années nous allons probablement assister au déploiement de plusieurs services, comme la VoD, le streaming en temps réel et les jeux en ligne, basés sur ces solutions P2P contrôlés.

En plus, les fournisseurs pourrons être intéressés par des solutions purement logiciels en pair-à-pair pour fournir des services à leurs clients quand ils sont hors de leur réseau.

Compte tenu de ces évolutions possibles, les travaux menés dans cette thèse pourront être utiles pour le déploiement d'applications streaming, que ce soit pour des milieux contrôlés ou non-contrôlés

Des applications streaming en temps réel performantes pourront être développées au moyen d'une approche mesh couplée avec des mécanismes d'incitation. Ces solutions pourront facilement intégrer des techniques *network and locality-aware* qui sont un sujet de recherche très populaire actuellement. D'autres études sont nécessaires pour améliorer ces techniques, et pour mieux comprendre l'interaction entre l'allocation des ressources au niveau réseau et au niveau applicatif. En outre, beaucoup d'applications streaming P2P sont conçues pour la diffusion d'un unique stream, et des études supplémentaires sont nécessaires pour permettre la diffusion de plusieurs streams simultanés sur le même overlay, et fournir un zapping (changement de chaîne) rapide.

Dans le cadre du streaming à-la-demande nous avons montré dans cette thèse, que l'allocation aléatoire des contenus et les mécanismes de mise en cache sont efficaces pour fournir le service. Toutefois, les techniques de connexion que nous avons proposées sont très structurées et risquent de supporter difficilement des environnements dynamiques. Des études complémentaires seront nécessaires pour coupler une allocation des contenus aléatoire avec caching à des approches mesh pour la diffusion.

Nous n'avons pas considéré les mécanismes pour la distribution pro-active des différentes copies des contenus à stocker dans des applications à-la-demande. Ceci n'est pas un problème si l'application cible est un service de VoD, où les contenus sont mis à jour de temps en temps. Toutefois, si l'application cible est un service de type UGC, où des milliers de contenus sont uploadés par les utilisateurs chaque jour, des techniques de mise a jour du catalogue rapides et efficaces sont nécessaires. Ces mécanismes peuvent être un sujet intéressant pour de futures études.

Le nombre croissant de réseaux overlay et l'apparition continue de nouvelles applications met en évidence le fait que l'intérêt des utilisateurs porte aujourd'hui directement sur les contenus et les services plutôt que sur les entités qui les stockent. Par contre, l'infrastructure réseau actuelle est entièrement conçue autour du principe end-to-end. Cette situation a conduit à une popularité croissante du concept *Content-Centric Networking* récemment promu par Van Jacobson. Dans cette approche, le réseau se concentre sur les données au lieu de leurs emplacements physiques. Compte tenu des tendances actuelles, cette solution est très attrayante pour le développement des futurs réseaux, et les utilisateurs et les fournisseurs de services peuvent en bénéficier. Certaines contributions de cette thèse, en particulier celles liées aux techniques de cache et de stockage dans le streaming à-la-demande, pourrons être utiles pour les travaux de recherche en cours sur le Content-Centric Networking.

Abstract

Multimedia streaming over the Internet strongly increased its popularity in last years. Media traffic has kept growing and is expected to increase tenfold within five years. To support such big audience, the traditional server-based architecture requires huge amount of storage, bandwidth and computational resources. This thesis considers streaming applications based on peer-to-peer architectures, which may overcome these resource constraints and cope with these evolutionary trends. In a peer-to-peer system the resources increase with the number of users; we show this is indeed effective to improve service scalability while reducing provider costs.

A first part of the thesis is devoted to the *mesh approach* to peer-to-peer live streaming, which has been used by popular commercial applications like PPLive and SoapCast. Through an experimental evaluation of PULSE, an unstructured peer-to-peer live streaming system we designed and developed, we analyze the streaming process in unstructured systems, and the impact of locality and resource awareness on their performance. To better understand the critical aspects of the stream dissemination in unstructured networks we focus on the building blocks of the diffusion process: the chunk/peer selection algorithms. We design and analyze some simple, yet practically interesting, selection policies for homogeneous and heterogeneous systems, and we consider the impact of system parameters such as the source selection policy, the chunk and neighborhood size. Our approach mixes theoretical results, when available, with empirical observations in order to give the best possible insights.

A second part of this thesis considers the on demand streaming, and in particular *server-free* approaches where clients collaborate to store and distribute contents to serve requests generated by other users. We analyze the size of the content catalog such kind of systems can provide to their users, as a function of storage and bandwidth constraints. By means of simulations and experimental evaluations, we analyze simple practical techniques that can be used for content storage and distribution in a fully peer-to-peer on-demand streaming system.

Key-words : content distribution, multimedia streaming, live streaming, on-demand streaming, peer-to-peer networks, distributed systems

Résumé

Le multimédia streaming sur Internet est devenu de plus en plus populaire ces dernières années. Le trafic média est en croissance continue et devrait décupler d'ici 2013. Pour soutenir cette large demande, les architectures centralisées traditionnelles, comme l'architecture client-serveur, requièrent énormément d'espace de stockage, de bande passante et de puissance de calcul. Pour surmonter ces contraintes de ressources et faire face aux tendances à venir, de nouvelles architectures doivent être mise en place. Dans cette thèse, nous étudions des applications streaming basées sur une architecture pair-à-pair. Dans un système pair- à-pair les ressources augmentent avec le nombre d'utilisateurs, ce qui permet le passage à l'échelle des services tout en réduisant les coûts pour les fournisseurs.

La première partie de la thèse est dédiée à l'utilisation de l'approche *mesh* (non-structurée) pour le streaming en temps réel pair-à-pair. Cette approche est utilisée par des applications commerciales populaires telles que PPLive, SoapCast etc... À partir d'une évaluation expérimentale de PULSE, un système pair-à-pair non structuré que nous avons conçu et développé, nous analysons le processus de diffusion en temps réel dans des systèmes mesh et l'impact de mécanismes qui prennent en compte la localité et les capacités des nœuds. Pour mieux comprendre les aspects essentiels de la diffusion en *streaming* dans les réseaux non structurés, nous nous concentrons sur les éléments de base du processus de diffusion: les algorithmes de sélection des chunks et des pairs. Nous proposons et analysons quelques politiques de sélection simples ayant un intérêt pratique pour les systèmes homogènes et hétérogènes, et nous estimons l'impact de paramètres tels que la politique spécifique de sélection de la source, la taille des chunks et celle du voisinage. Notre approche mélange des résultats théoriques, lorsqu'ils sont disponibles, à des observations empiriques pour donner les meilleures intuitions possibles.

La deuxième partie de cette thèse considère le streaming à-la-demande, et en particulier des architectures sans serveur, où les clients collaborent pour stocker des contenus et les distribuer au fil des requêtes générées par les autres utilisateurs. Nous analysons la taille du catalogue de contenus que ce type de systèmes peut fournir à leurs utilisateurs, en fonction des capacités de stockage et de bande passante. Nous considérons ensuite comment de simples techniques pratiques peuvent être utilisées pour réaliser effectivement de tels systèmes. Nos résultats proviennent à la fois d'une analyse théorique, de simulations et évaluations expérimentales.

Mots-clefs : distribution de contenu, streaming multimédia, en temps réel, et à-la-demande, réseaux pair-à-pair, systèmes distribués