UNIVERSITE DE NICE-SOPHIA ANTIPOLIS

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

T H E S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention : Informatique

présentée et soutenue par

Fabio PIANESE

Systèmes Pair à Pair Pour la Diffusion de Données Vidéo

PULSE - Un Système Adaptatif pour le Streaming en Direct sur Internet

soutenue le *3 Décembre 2007*

Jury :

Dr. Walid DABBOUS	Président
Prof. Dr. Ernst BIERSACK	Directeur de Thèse
Prof. Dr. Wolfgang EFFELSBERG	Rapporteur
Prof. Dr. Pascal FELBER	Rapporteur
Dr. Joaquín KELLER	Examinateur

PULSE
An Adaptive Practical Live Streaming System

(PULSE - un Système Adaptatif pour le Streaming en Direct sur Internet)

Fabio Pianese

October 31st, 2007

Final Version 1.0 - January 31st, 2008 - Library Version

Cum sit enim proprium
viro sapienti
supra petram ponere
sedem fundamenti
stultus ego comparor
fluvio labenti
sub eodem tramite
numquam permanenti

– *Carmina Burana*

Abstract

Live Streaming consists in distributing live media (video and audio) to large audiences over a computer network. Providing a live streaming service over the Internet presents many challenges: the application must respect the timing and quality constraints imposed by the nature of live media and by user expectations while struggling with the practical limitations due to the *best effort* properties and unpredictable dynamics of the Internet. Because of the limited deployment of native IP multicast, an Internet-based live streaming application with a global scope can only rely on *end-to-end* network primitives, such as unicast connections. The traditional client-server approach to live streaming has a serious scalability limit, as the upload capacity requirement at the server grows linearly with the user population. A P2P solution has the big advantage of seamlessly scaling to arbitrary population sizes, as every node that receives the video, while consuming resources, can at the same time offer its own upload bandwidth to serve other nodes. In theory, if every node contributed on average at least as much as it consumed, the P2P system would have enough resources to grow indefinitely.

This work presents and evaluates PULSE, a practical P2P live streaming system intended for large-scale deployment over the Internet. PULSE uses an unstructured mesh-based design and relies on local pairwise incentives as its peer selection mechanism. The most innovative feature of PULSE is the unique coupling of incentives with feedback derived from data reception, which leads to the emergence of clusters that regroup nodes with similar resources. By exploiting this intrinsic clustering phenomenon and by leveraging latency measurements to estimate network locality, PULSE is capable to successfully operate in a wide range of resource-constrained real world scenarios and to support dynamic user populations and heterogeneous node upload capacities.

Résumé

Le *live streaming* consiste en la distribution d'un flux de données multimédias (vidéo et audio) en direct vers une large audience par le biais d'un réseau d'ordinateurs. Offrir un service de streaming live sur l'Internet présente plusieurs défis: respecter des contraintes de délai et de qualité, imposées par la nature des données audiovisuelles, en utilisant un réseau *best effort* au comportement aléatoire. Par ailleurs, le support pour le multicast IP n'a pas été déployé de façon suffisante, ce qui oblige toute application qui vise un déployement sur échelle globale à utiliser des connexions *de bout en bout*. L'approche traditionnelle client-serveur souffre d'un goulot d'étranglement au niveau de la bande passante remontante du serveur, dont la consommation augmente de façon linéaire avec le nombre d'usagers. Une approche pair-à-pair (P2P) a l'énorme avantage de permettre le passage à l'échelle du système jusqu'à une taille arbitraire, puisque chaque nœud qui reçoit le flux vidéo peut apporter ses ressources au réseau au même temps qu'il en consomme. En theorie, au cas où chaque nœud apporterait en moyenne la même quantité d'upload qu'il consomme, le système pourrait s'accroître indéfiniment.

Ce travail présente et évalue les performances de PULSE, un système de streaming en direct adapté aux exigences pratiques d'un déploiement à large échelle sur l'Internet. PULSE utilise un réseau maillé non structuré (*unstructured mesh*) et applique des mécanismes locaux d'incitation au partage en tant que critères pour le choix des nœuds avec qui établir des associations. L'aspect le plus innovant de PULSE est l'introduction d'un mécanisme qui combine l'incitation avec une boucle de rétroaction basée sur les délais de réception, qui mène à la formation de clusters parmi les nœuds qui partagent la même quantité de ressources. Exploitant cette organisation émergente et utilisant des mesures de latence de bout en bout pour estimer la proximité entre nœuds, PULSE peut supporter un large éventail de scénarios réels où les ressources sont rares et distribuées de façon hétérogène, avec une large population d'usagers au comportement dynamique.

Acknowledgments

The last time I had the opportunity to compile a list of people that I felt deserved recognition [84] I ended up with two pages full of names, in-jokes, and fond memories. This time I will be more sober and to the point - mainly cutting down on the in-jokes - but still strive not to forget too many people.

Thanks to my family. It's been a long time since I left you to follow this unusual path - almost five years, 18% of my whole current lifetime, give or take. Yet, I constantly feel your love and support, and hope you feel mine, as strong as ever. Wherever I have been, wherever I will be, I wish you to know that I have ever been - and ever will be - grateful to you: for all that you have been to me, for what I am, thanks! I owe you everything.

I spent the first two years of my thesis in Paris. A wonderful place, a city I deeply love: I love your streets and boulevards, your gardens, your bridges, your monuments. I love your immortal spirit, this strange feeling of being in the heart of history and humankind. I love your restaurants, too! Maybe one day I will have the chance to live there again...

Thanks to my friends in Paris. It has been a pleasure to meet you, to hang out with some of you, to drink fine wine with some others of you, to have fun with everyone of you. Let's celebrate the wine-drinkers! I want to especially thank Thierry Baudoin: not only is he a great friend, he is also the one who provided some of the best wine bottles I have ever tasted. Thanks to Nicolas, my roommate in Paris for more than two years. How do I miss our old apartment in the 15th arrondissement! I wish you both well and to your families (and new-born children!). And *bonne chance* for you life in China, *ma poule*!

Thanks to my colleagues in France Telecom R&D, Issy: the time I spent with you has been so precious to me. I'm most grateful to Dr. Joaquín Keller for his guidance, support, and friendship over these years. Thanks to Dr. Gwendal Simon, it has been a pleasure to work with you. Also thanks to Dr. Frederic Dang Ngoc, the third member of our former NETOFPEERS team, I wish you the best for your North-American adventure. I feel that, as a ridiculously small team (of which I was the fourth and - unfortunately - last member), we have achieved a lot, and am proud of our work. If perhaps not in the right place, we have been doing the right things at the right time, and as well as we could do.

Thanks to my interns! It is with emotion that I acknowledge the role of Diego Perino, my first student, a person who has contributed a lot to my work, and a good friend. Diego wrote the code of the prototype PULSE node and helped with the testbed and PlanetLab measurements. Thanks for your tireless devotion to the task: you went well beyond your (not-so-simple) duty of intern, and I hope that the results we could get have fully paid your effort back.

Thanks to all the people I met in France Telecom R&D, to name just a few: my friend and fellow PhD student Franck Shen, Olivier Gachignard, Olivier Bouté, Dr. Fabien Mathieu, Dr. Patrick Brown, Jean Béhue. Many thanks to my secretary in Paris, Chantale Couvert.

I then had to leave Paris, and spent my last year in Sophia Antipolis between the Institut Eurecom and the local branch of France Telecom R&D. While I had to mourn the loss of my Parisian social life, which - to be honest - was replaced by no social life at all, I had the opportunity to work in close contact with many brilliant people. I wish to thank some of them here: Prof. Pietro Michiardi, Prof. Guillaume Urvoy-Keller, Dr. Damiano Carra, Dr. Taoufik en-Najjary, Moritz Steiner, Matteo Varvello, Daniele Croce. Thanks to Krishna K. Ramachandran ("ça va?"), it was really a pleasure to meet you here. Also, many thanks to Eurecom director, Dr. Ulrich Finger, and all of the Eurecom staff.

No social life, however, does not mean no human interaction at all. Even if remotely, I could count on the support of my dear old friends. They say: "years pass, things change". I firmly believe that most people do not. At least, you didn't, and I am thankful for that.

And here I am. These three years have been long, and I am both glad and sad as I approach the end of this venture. Of all the outstanding people who supported me during my work, I want to sincerely thank my advisor, Prof. Ernst W. Biersack. Even when in Paris, 900 Km away from your office, I have always felt backed by your knowledge, wisdom, and human support. I remember with fondness our evening meetings, held in Paris each time you happened to spend some time there, when we would discuss our work in unlikely places such as French *bistrots* or Scottish pubs... Thanks for teaching me how to do research. Thanks for allowing me the freedom I needed to manage my thesis project. Thanks for understanding the delicate compromise I had to strike between the academic and corporate worlds. Thanks for being there when I needed you, with advice, criticism, and praise.

This dissertation is the final result of three years of speculation, hypoteses, and experiments about a practical system that targets a very specific problem. Also, it is an attempt to look in a comprehensive way to the problem itself, from its historical roots to its current incarnation, and to build a systematic framework of the existing approaches to solve it. I hope that the reader will find this work - if not enjoyable - at least interesting and maybe useful. And I fully claim my responsibility for any mistake or inaccuracy hereby contained.

Saturday, July 14th 2007

Fabio Pianese

Contents

List of Figures

List of Tables

List of Algorithms

List of Main Variables

U_i	Upload capacity of the access link at node i
D_i	Download capacity of the access link at node i
$B_{i,j}$	Connection bandwidth between node i and node j
RTT	Network round-trip time
N	Number of nodes in the system
P_i	Node i (uniquely identified)
d	Node degree
SBR	Stream bit-rate [*Kbit/s*]
R	Chunk rate [*chunks/s*]
c_j	j-th chunk in the stream
T_B	Average node lag
T_V	Node playout delay
W	Size of node Sliding Window [*chunks*]
TW	Size of node Trading Window [*chunks*]
EPOCH	Time between two subsequent peer selections
C	Latency bias coefficient for peer selection
H_i	Local value of History score about node P_i

Chapter 1

Introduction

1.1 Why Peer-to-Peer Live Streaming?

We decided to concentrate our attention on the problem of live streaming for several reasons. First, scalable data distribution is a fundamental need in the Internet today: while the distribution of static (i.e. pre-stored) content has been a topic of widespread interest during the last decade, the attention to live content distribution has been more recent. The rise of peer-to-peer (P2P) architectures in the context of static content distribution has provided a definite improvement in scalability over the previous server-based architectures. Extending a P2P approach to live media streaming allows to address the scalability issues of centralized systems, presents interesting challenges, and still constitutes an open research problem.

Second, the understanding of live media distribution is much less consolidated than the classic case of static file distribution, as it is subject to timing and ordering constraints. Third, live streaming is a problem with clear boundaries and requirements: as new data are constantly generated *(i)* all the viewers are loosely synchronized in receiving roughly the same stream segment, *(ii)* the system as a whole has a short memory and *(iii)* it can operate with a larger independence on the user behavior patterns than bulk data distribution or VoD. Fourth, scalable live streaming applications have today a huge practical interest, as the equipment required (powerful computers, digital cameras or web-cams, Internet connections) is available off-the-shelf and is usually cheap: this constitutes a fundamental premise for the commercial success of a new application or service, enabling it to spread and succeed among the general population. Finally, we believe that there are several aspects of the existing practical P2P live streaming systems that can be improved, and we make several contributions in order to do so.

1.1.1 A Potential 'Killer Application'

In the beginnings of the Internet, the only type of data that could be exchanged and displayed by users was text. Images were then introduced: initially in the form of large uncompressed bitmaps, then compressed using formats with an ever-increasing efficiency since the beginning of the 90's. The continuous growth in the access bandwidth and CPU power has allowed to integrate and display complex combinations of text and images in the form of WWW pages. Web-aware scripting (Javascript) and programming languages (Java, ActiveX, Flash) have added an interactive dimension to otherwise 'static' web pages. Full applications that are able to run inside the ubiquitous web browsers allow today to perform tasks that once required the installation of platform-specific binary executables. The seamless diffusion and reception of live media streams to arbitrarily large audiences could well be the next successful killer application of Internet technology.

A Long-Awaited Development For a long time, video over the Internet has actually been considered by many "the next big thing". During the last 20 years, Internet service providers and telephone companies have devoted exceptional resources to the study of the impact of video distribution over their networks, both in technological terms (how to transport the media data to the home users, how to guarantee appropriate levels of quality) and from an economical perspective (how can we make money from it). The research in networking has followed the expected needs of the industry, providing insights on the theoretical feasibility of the transmission of video and on the technological limitations that had to be overcome to allow an efficient distribution of data from one source to many destinations.

The Shadow of Illegality Quite recently, a formidable interest for streaming technology has also been expressed by the well-established content providers, along with equally formidable concerns about how to protect their valuable content, control its distribution, and avoid to endanger their existing market. Having realized the inherent danger of the free flow of information allowed by the Internet to their current business model, they have set in motion a titanic effort to protect their interests by way of technical measures [6], legal litigation [2], propagation of FUD (fear, uncertainty and doubt) over the press and the media, and lobbying efforts toward lawmakers and regulatory bodies.

The success of this multi-pronged effort menaces the success of large-scale media distribution applications. The danger comes especially from the legislative and regulatory fronts: the enactment of laws that set arbitrary limits on purely technological matters, or the introduction into hardware and software standards of technical measures that prevent the "unauthorized" use of the capabilities of general-purpose computing equipment. While in the regulatory front the manufacturers have shown to be hostile to restrictions mandated by standards, as they decrease the usefulness of the products and thus their value for their customers, there is no such market pressure on the lawmakers. Several countries have been adopting laws that not only sanction copyright infringement as a civil offense, but that punish as penal offenses the *circumvention of technical restrictions* (e.g. Digital Millennium Copyright Act [7] in the USA, European Union Copyright Directive [1] in Europe) or the conception use and distribution of peer-to-peer applications (e.g. the "Droits d'Auteur et Droits Voisins dans la Société de l'Information" law [4] in France), and assimilate to theft the fact of downloading copyrighted data (e.g. Italian and

German law).

A solution to this heated controversy between the owners of popular content and the general public does not seem to be in sight yet, as it would probably require a full rethinking of established juridic concepts such as *copyright* and *intellectual property*.

1.1.2 Beyond the Limits of a Centralized Approach

It is quite interesting to notice that the distribution of media over the Internet has not taken off thanks to the explicit intervention of the industrial or academic world. Rather, the small players - such as single individuals, start-up ventures, and technically educated Internet users as a whole - have often been the first to explore the possibilities of the existing network infrastructure, anticipating the so-called *innovations* later introduced by the established players of the telecommunications field.

The development and success of P2P systems is as a good example of user-driven innovation. These small players were in fact the ones confronted with serious practical issues, such as lack of economic resources, unreliable hardware and limited connectivity - they could not afford powerful server machines, nor large data storage facilities, nor fast Internet access. For them, P2P was more a necessary evolution than an incremental optimization over the server-based architectures.

In a world where servers were required to support every kind of online activity - from data download, to search, to instant messaging, etc. - the adoption of a distributed approach that enabled the exploitation of the resources provided by the users was a definite breakthrough. Initially, there was widespread skepticism over the effectiveness of P2P networks as a replacement of server-based technology. Especially the earliest P2P software such as the first iterations of Gnutella were not meant to support large scale systems [91], and were expected to be deployed in local contexts with small user populations.

The astonishing growth in popularity of these applications, coupled to the reduced need for resources at the “server”, quickly spurred a considerable interest. Suddenly, P2P became a sort of *buzzword*, which was supposed to grant either the immediate success of a new application, enormous cost savings at the service provider, or both. This phenomenon sometimes reached unreasonable and hilarious proportions (“*Let’s do <anything> using P2P! Quick!*”) before finally giving way to more rational behaviors.

Today, the design of large-scale services and applications benefits from a large body of lessons learned from the evolution of P2P applications. The most important insight is that a P2P approach works well to solve few specific issues but is impractical for some others. Problems such as large-scale data distribution [33], data storage, keyword-based or pattern-matching search [73] can enjoy significant benefits when distributed techniques are adopted. On the other hand, features such as user presence tracking and authentication are extremely challenging to reproduce in a pure P2P fashion.

1.1.3 Advantages of P2P Live Streaming

Live streaming is a challenging application because of its timing constraints. However, it is also particularly interesting because these constraints are loose enough to leave a significant degree of flexibility in the architectural choices available to the system designer. While few tens of seconds are allowed between the generation of data at the source and the reproduction at the receivers, this short time frame can still be sufficient to process the data stream and to distribute it through multiple “hops” over the network. *Thanks to the non-negligible amount of tolerance to playout delay, the live streaming application allows the adoption of a large range of technical solutions, including P2P techniques.*

A P2P approach has the big advantage of making the system **scale seamlessly to arbitrary sizes**: every node in the P2P network, while consuming system resources, will at the same time offer its own resources to serve other nodes. In principle, if every node contributes at least as much as it consumes, the P2P system will be able to grow indefinitely. This property, called *self-scalability*, is the first compelling reason for the research in P2P live streaming.

Moreover, in a client-server architecture the cost of media streaming is completely bestowed on the content provider, as it requires the allocation of an amount of resources at the server side directly proportional to the expected peak audience. The media source in a P2P streaming system just needs a constant (and small) pool of resources, as each user can contribute to the system. The **economic benefits** of this property are especially relevant for large user populations.

1.1.4 Challenges in P2P Live Streaming

In recent years, a number of P2P systems for live streaming, which adopt several different architectures, have been proposed. While Chapter 2 will be devoted to the analysis and comparison of the systems in the literature, we anticipate here the most relevant issues that are driving the research activity in this field:

Scalability While resources may scale without limits, a theoretical upper bound of the system size exists for time-sensitive applications such as live streaming. This limit is due to the delay introduced by each “hop” that is traversed by the media data. As a consequence, the primary challenge of P2P live streaming is to devise a data distribution technique that guarantees the **constant propagation** of the media data through the system and **avoids an excessive delay build-up** as the user population increases.

Bandwidth Awareness Many practical issues arise when switching from the classic client-server paradigm toward a P2P approach. Another challenge to achieving self-scalability comes from the questionable assumption that all nodes will contribute sufficiently to the system. Indeed, nodes may contribute *less than what they consume*, or even *nothing at all*, either because of inherent technical limitations (lack of resources, presence of a firewall) or on purpose, having made the explicit choice to defect (*freeloading*).

Fairness If the assumption of cooperation does not hold, several problems (such as the rejection of new users or the loss of data) can compromise the functionality of the system and the playback quality. This problem can be approached from two main points of view: as a matter of **resource allocation**, *i.e.* how to place the peers in the system and distribute the data in order to to efficiently exploit the available resources, and of **incentivation** (or access control), *i.e.* how to discourage or ban from the system the nodes that do not contribute enough.

Locality Awareness Another widespread concern in P2P content distribution comes from the fact that the data are replicated in a non-optimal way compared to native IP multicast, as the application is not aware of the underlying network topology. The lack of locality awareness, namely higher media reception delay and redundant link utilization in the network, can degrade the performance of time-sensitive applications such as live streaming.

Resilience In addition to the best-effort and unreliable nature of the network, using intermediate nodes as functional elements of the system increases the probability that the service will be disrupted by transient changes in the system membership. Since a large-scale P2P streaming application relies almost completely on ordinary users to provide the service, the original media provider has little control on the way data is distributed and on the quality perceived by the users. Unlike the underlying network infrastructure (i.e. routers, cables, etc.), which has an extremely high availability and fault tolerance, the applicative overlay built by the nodes of a P2P system (i.e. software processes running on user hardware) offers no such guarantees [14]. Any peer can be expected to join the system, leave the system, misbehave, or fail at any moment: to provide an uninterrupted service, the system must hide the effects of node transience and manage to restore as soon as possible its correct functionality.

Accessibility The ease and freedom of access by potential broadcasters to the live streaming infrastructure can be seen as a further technical challenge in system design: the application should not require a large amount of resources (upload capacity) at the media provider, so that users with broadband Internet access can still act as streaming sources. The application should be easy to set-up and use, so that casual users will not be discouraged from participating in the system. In a system where the content is generated by the users, accessibility is the key factor that boosts the user-perceived value the application.

1.2 Issues Not Covered in This Thesis

There are several related issues that we do not investigate in this thesis:

- *Security Issues*: Security is a fundamental aspect in every system, especially when large-scale computer networks are involved. Besides the problem of protecting the integrity of the media data, which is briefly touched on in Chapter 3, and the need to protect the system from users who do not contribute sufficiently to the system, which is a central aspect of this thesis, all other security issues, such as robustness to denial of service (DoS) attacks and to blatantly malicious behavior (*byzantine* peers, *sybil* attacks [39], collusion between nodes, manipulation of protocol information, etc.), lie out of the scope of this work. Due to the partial reliance on altruism to improve and stabilize the system performance, we do not guarantee either that the algorithms and protocols presented in this thesis are capable to withstand strategic exploitation by greedy players [88].

- *Optimality of Chunk Selection*: Peer selection and chunk selection algorithms are the two core elements of data-driven P2P live streaming systems. In this thesis we focus on scenarios where nodes are non cooperative and their capacities are highly heterogeneous, as required by our practical approach: we argue that, in these circumstances, peer selection is the most important algorithmic component and, as such, it deserves to be the primary focus of our attention. Our choice of chunk selection algorithm, while carefully reasoned, was largely based on empiric considerations.

- *Comparison with Other Systems*: An evaluation of PULSE against live streaming systems in the literature is not included in this work. We are convinced of the usefulness of this study, as it is the logical fulfillment of a practical study: *"Will it work better or worse?"* Unfortunately, as far as we are aware, there is a sore lack of published experimental results or data traces from other data-driven systems that can be used as a basis for comparison (i.e. verifiable and obtained under reproducible conditions). Our motivation in releasing the prototype code and publicly documenting the algorithms is to provide to the community a reference for future comparisons.

1.3 Thesis Contributions

This thesis makes several contributions. It first approaches the problem of **live media streaming from a practical point of view**. The requirements of large-scale P2P streaming applications are presented and discussed starting from the **set of challenges** enumerated above, which reflect the current technical limitations of Internet technology (and their likely evolution in the foreseeable future). An in-depth survey of the related work is then performed, whose aim is to evaluate the existing live streaming architectures in light of their suitability to a large-scale deployment over the Internet. Insights on the architectural features that help toward this goal are also provided.

The second contribution is the **design of PULSE**, a P2P live streaming system that satisfies the previous requirements. The PULSE system is among the first systems to rely on an unstructured mesh-based design, and introduces an incentive-based mechanisms for the selection of neighbor nodes. By leveraging an intrinsic resource-based clustering (due to the pairwise incentives) and by exploiting the loose synchronization between peers (using a feedback mechanism based on reception performance), PULSE is capable to operate in a wide range of real-world scenarios. The advantages of PULSE over existing systems can be summarized as:

- Support for very high levels of churn (node arrivals and departures)
- Support for strongly heterogeneous distributions of peer upload capacity
- Efficient use of the available upload capacity, especially under scarcity of resources
- Fast adaptation and recovery from abrupt changes in network conditions
- Implicit awareness to network locality through latency measurements
- Attention to the quality of media playback, striving to minimize degradation

The third contribution is a comprehensive set of **metrics for generic data-driven systems**, complemented by additional metrics that are useful to assess the bandwidth and latency awareness of adaptive live streaming systems such as PULSE. A small survey on the current techniques that allow to describe the behavior of data-driven systems, covering both theoretical models and empiric methods, is also included.

The fourth contribution is the **implementation** of a **simulator** that models the complex behavior of a PULSE system. Based on the insights obtained by experimenting with the simulated algorithms, a stand-alone **node prototype** was also implemented. These pieces of software have been used to improve our understanding of the emergent global behavior of PULSE systems that operate under a variety of bandwidth distribution scenarios, node membership patterns, and network environments.

The fifth and last contribution is the qualitative and quantitative **analysis of PULSE** based on simulation and emulation results. We first validate that the PULSE algorithms are operating as expected, and then assess their performance in a large range of challenging scenarios in which structured systems would hardly be able to operate. We specifically evaluate the awareness of the resulting overlay mesh to resource availability in the system and to pairwise network latency, and describe the average characteristics of the data paths that connect the source to the nodes.

1.4 Thesis Organization

The rest of the thesis is structured as follows. Chapter 2 presents live streaming over the Internet in an historical and technical perspective. The current solutions for P2P live streaming are then introduced: after restating in better detail the main challenges of this problem, the various available design options are compared and conclusions on their viability are drawn. Chapter 3 describes the PULSE system in its entirety - the basic insights, the terminology, the algorithms, and the implementation. Chapter 4 approaches the problem of understanding a dynamic mesh-based system like PULSE: after a survey on the most recent theoretical models and on their current limits, we illustrate the empirical techniques that we had to adopt to study our system. Chapter 5 introduces an original set of performance metrics to describe the behavior of a generic data-driven system. An additional set of metrics that correlate the availability of node capacity and data reception performance is defined: these metrics will be the fundamental tools of our subsequent analysis of PULSE by way of simulation in Chapter 6 and emulation on medium-scale network testbeds in Chapter 7. Chapter 8 concludes this work.

Chapter 2

Related Work

This chapter approaches the subject of live streaming in a broad perspective. Section 2.1 briefly traces the evolution of modern-day streaming from its early roots in radio broadcasting. Section 2.2, presents the requirements and the main challenges of supporting large scale data distribution under timing constraints. Section 2.3 follows the chronological evolution of Internet-based streaming systems during the last decade, with an analysis of the properties of the main overlay architectures and a study of their practical advantages and limitations. Section 2.4 concludes the chapter.

2.1 A Brief History of P2P Live Streaming

Human beings have a distinctive trait in that they can learn from other people's experience[1]. Communication is the principal vector through which information is conveyed to others. The human society can be seen on many levels as a "communication framework", which allows the replication and transmission of knowledge both across time (e.g. from parents to children) and across space (e.g. among individuals of the same population).

Looking back in history, we can notice how the means used to spread knowledge to larger and larger audiences have closely followed the growth of the human population. Also, we can appreciate how the available means of communication have been at all times functional to the structure and the needs of the predominant social organization.

The demographical increase of the human race, together with the establishment of dense communities living in limited spaces, has introduced the need for new methods to effectively deliver information to people. The long way from the early beginnings of human organization - when men supposedly began to aggregate in small groups of nomadic hunters-gatherers - to the current globalized post-industrial mass society has been paved by technological breakthroughs, which have then become the necessary basis for the subsequent social evolution. We can think about

[1] "[and] are also remarkable for their apparent disinclination to do so." (Douglas Adams)

carved signs on the stone and the invention of alphabets; the evolution of writing supports, from clay tablets to parchment, papyrus, and then paper; the use of scrolls, replaced by codices and books; the printing press, which made possible the diffusion of literary works on a broader scale, such as pamphlets and newspapers; the mechanical printing machines, capable to generate thousands of copies of a single text in a short time.

In the last two centuries, the technological breakthroughs related to human communication have occurred at an ever increasing pace, thanks to our improved scientific understanding of physical phenomena: the discovery of the laws of electricity and electro-magnetism, the telegraph, the telephone system, the birth of radio communications, the development of computing machines, the advent of long-distance computer networks, the tremendous advances in digital electronics and optoelectronics. All these technologies distinctively shape the evolution of the human society, revolutionizing in large part the way people live, behave, think, and perceive the reality.

The 20^{th} century has been profoundly influenced by the advent of the first two forms of real-time mass-media: radio and television broadcasting. Suddenly, the horizon of the knowledge available to common people, previously limited to a small geographical scope, was expanded to a national or continental scale, and the speed at which information could spread among the population became nearly instantaneous. The public opinion was thus born. The art of harnessing the power of the masses also began to develop. Information and misinformation started to become tactical weapons, called intelligence and counter-intelligence. Wars were fought, won or lost with the help of AM radio waves and TV shows. Political speeches, news, ideology, moral values, fashion, religion, social trends, education, advertisements - all these (and more!) started flowing each day into every household by the small TV screen.

Television does not only allow ’to see far’, as its name implies. It creates and distributes popularity, it forges a background of well-known concepts, and facts that will be shared by people over an enormous spatial scale. Radio and television have now become an unprecedented source of *human synchronization*, as they regularly dictate, through their schedule and message, a single and common dimension of time and culture.

2.1.1 Live Streaming, or the Internet Television

An inherent property (and limitation) of radio and television is that they are a one-to-many broadcasting schemes. The access to these technologies by private citizens is restricted by many factors: the cost of the equipment necessary to produce, process, and transmit information over a meaningful distance, the regulatory rules, the limited availability of licensed radio frequency bands, and the difficulty of gaining popularity starting from a complete anonymity.

Cable Television and Direct-to-Home satellite television have begun to emerge in America and Europe in the 1970s, offering a much larger choice of television channels and service providers. They did so using either a wired medium (fiber/coaxial cable) or microwave frequency channels (C band around 4 GHz, K_u band between 12 and 18 GHz), which are subject to less stringent regulations. The recent and widespread push for digital terrestrial television broadcasting will probably further mitigate the limitations due to the scarcity of UHF/VHF frequency bands. However, the use of different frequencies and signal encodings does not eliminate any of the fundamental restrictions on the open access to the medium.

The availability of high-speed computer networks and powerful commodity hardware has opened new possibilities for the distribution of video and audio data. If we transcend the classical implementation of television, which uses a closely-regulated medium to convey information, we can easily imagine a system where another medium of communication, for example a public computer network, is used to replace the radio channel.

The Internet: An End-to-End Paradigm The advent of the Internet as an open and vendor-neutral world-wide communication infrastructure has deeply marked the end of the last century. It is quite interesting that the enormous growth in size and importance of the Internet initially went largely unnoticed by the masses, until around 1996, when the "Internet phenomenon" suddenly emerged: in a few years, the general population became aware of the Internet, that started to be a prominent factor in the culture and lifestyle of most first-world countries.

The evolution of Internet applications has closely followed the technological development of the medium: from the earliest program, *e-mail*, which was intended to deliver short text messages between users of different machines - followed by *ftp*, to perform reliable file transfers - to the *world wide web* (in origin just HTTP+HTML), for the retrieval and presentation of interlinked structured data, and to a plethora of other applications that deal with new data typologies and address specific user needs. This evolution has only been possible because of the basic technical and philosophical foundation of the Internet: the *end-to-end principle*.

On the Internet, packet switching supported by a routing protocol is used to propagate the data over the network. Data are cut into small segments called *IP datagrams*, and source and destination addresses are associated to the individual datagrams before they are sent out into the network; the core of the network is made of routers, computers whose main task is to send the datagrams hop after hop toward the correct destination, making forwarding decisions based on the content of their routing tables. A small number of *transport protocols* offer the basic functionalities required by the majority of the applications, such as reliable, ordered, at most once data delivery (TCP) and unreliable data delivery (UDP).

The Internet is built so that the network itself is basically *dumb*, as it only provides a way for information to reach its final destination. The *intelligence* of the network only resides at the end points, where applications are run by the users and deal with the users' data. The network does not cares neither about the data format, nor about the protocol that an application uses. The end-to-end principle has given a precious freedom from technical constraints to Internet users, and it has made (and still makes) it possible to deploy and test new applications without the need to make modifications *inside* the network, and with no supervision or regulation from any entity.

A Fast-paced Technical Evolution The first thoughts about real-time distribution of live data as a major Internet application are certainly very old, probably dating back to the early vision by its founders. The introduction of IP multicast, in the middle of the ’80s, is already motivated in part by the practical need of streaming applications: in RFC 966 (December 1985) we can in fact read that multicast can be *“useful to several applications, including* [...] *conferencing*”, and that one of the strengths of multicast over unicast is that “*transmitting multiple* [unicast] *copies of the same packet makes inefficient use of network bandwidth, gateway resources and sender resources. For instance, the same packet may repeatedly traverse the same network links and pass through the same gateways*”. These early mentions of “streaming” applications were mostly speculative at the time, as many other technical requirements to the transmission of video and audio on the Internet were still missing.

The Internet, in fact, has become a viable medium to transmit near-real-time audio and video data only over the last decade. This was due to the concomitant developments in several technical and social domains: the development of powerful and affordable personal computers and media acquisition hardware, the advances in the video/audio coding algorithms and standards, and the widespread improvement in speed and in affordability of the default commercial Internet access technology.

The hardware was one of the first limitations to be overcome: CPUs started to be powerful enough to decode an MP3 stream only around 1995, but the improvement in CPU designs (superscalar architecture, branch prediction, multimedia extensions, integrated vector computing, multiple cores, etc.) and building technology (constant transistor miniaturization, steady increase of clock frequency, improvement in semiconductor quality and design) was then so fast that high-quality video playback became possible on high-end machines since around 2000. Coding algorithms also saw huge improvements during the same time frame. Improving on the MPEG-1 specification (1988), which required a 1.2 Mbit/s bitrate for a television-quality stream, the subsequent MPEG-2 (1994) and MPEG-4 (1998-2005) standards focused on better compression and subjective quality[2].

Meanwhile, the Internet was undergoing a rapid evolution from a non-commercial network used for research and educational purposes to its current “information superhighway” status. After the ban on commercial activities was lifted (mid-1994), commercial Internet service providers (ISPs) began to develop their access networks, gradually reaching a pervasive coverage of the world’s richest countries. Internet began to be known outside of the academic circles, and started to gain an ever-increasing role on the day-to-day life of many people, both for work and leisure purposes.

The feasibility of live streaming over the Internet was initially limited by the small access bandwidth that was available to most users: ten years ago dialup connections at a nominal rate of 28 to 56 Kbps were still the most common access technology, barely enough to send or receive a single media stream of low quality. The advent of private cable networks, of the DSL family of access technologies, and more recently of *fiber to the home* (FTTH) has solved the problem of download bottleneck on the users’ side. Today, the ISPs of most countries provide connectivity at downstream speeds between 2 and 20 Mbps.

[2]Today, a 500 Kbit/s stream is commonly rated as television-quality for all practical purposes.

2.1.2 The Long Wait for Native Multicast

As we said above, the end-to-end paradigm around which the Internet was built has been critical to its success. Because of this paradigm, many new end-to-end protocols were created, some of which succeeded and became the standard protocols still in wide use today. On the other hand, the inherent limitation of the end-to-end paradigm is that the number of ends involved in each protocol transaction is two: each node on the Internet has a name and can reach (and be reached by) any other node[3], but no *group* interactions (e.g. point-to-multipoint, multipoint-to-multipoint) are made possible by the IP network layer.

End-to-end worked very well in supporting the ’connection’ metaphor, a direct link between named ends, an old idea that was deeply rooted in the legacy of the telephone. But some applications do not need named ends, nor single pairwise connections: this is the typical case of *data diffusion applications*, where the focus is not on who provides the data, but on the data themselves. This class of applications includes live media streaming, video-on-demand streaming, and bulk content distribution, which all suffer from the network-layer limitation to point-to-point connections. If it were someway possible to synchronize the state of more than two peers that are interested in retrieving the same information, and if the underlying physical resources could be allocated in an appropriate way, then a ’group communication facility’ could be introduced at network level to provide seamless support for data diffusion applications.

The thoughts of the proponents of native IP multicast in the mid 1980s went probably along those lines. A multicast network primitive implemented at each router would have been the optimal solution to the diffusion problem in terms of backbone link utilization, since at that layer there is full visibility on the local links and on the global routing tables. Advantages could have been significant also on the limited scope of individual domains, especially for the efficient distribution of locally-relevant information to the customers of a same ISP, the students of a same university, etc.

A large amount of research on IP multicast has been conducted from 1985 to 2000. By 2000, most of the technical issues had been sorted out [17], such as scalable algorithms for reliable data dissemination [43][79] and multicast routing [35]. The problems that could not (and will never) be solved were mainly in the fields of multicast congestion control ([22] proved that the maximum throughput of a one-to-many transmission with *coupled* congestion control decreases with the logarithm of the number of nodes in the multicast group) and of multicast group management.

It is a common opinion [38] that the native IP multicast infrastructure failed to be widely deployed because of several concurrent reasons, including:

- the standardization by IETF of an open, unmanaged multicast infrastructure composed by a multitude of alternative protocols that covered the same functionalities;
- many security concerns, especially about the resistance to attacks targeting the multicast routing infrastructure, the ability to guarantee the integrity of the distributed data, and the

[3]Actually, this is no longer the case today, as the introduction of NAT and other packet-mangling *middle-boxes* has had the side effect of disrupting the original *perfect* end-to-end connectivity.

issue of controlling of the actual scope of a multicast session (e.g. by authenticating its participants and by performing access control in a distributed manner);

- probably the most important reason, the lack of a commercial interest by ISPs in investing money to provide a native multicast facility when popular applications that require multicast do not yet exist.

We agree with the statement that most ISPs are not enabling inter-domain multicast routing likely because the bandwidth savings they can expect are far outweighted by the prospective costs of deploying and supporting the multicast functionality (despite the hardware support for IP multicast is available out-of-the-box in recent network equipment and operating systems). Few ISPs support the full inter-domain multicast specification, whose scope is however limited by the fact that most other ISPs do not support it. Some ISPs are known to just implement intra-domain multicast (e.g. Orange France), so the scope of multicast sessions is limited to the customers of the same ISP.

We conclude that, today, the deployment of native IP multicast is not adequate to make it a viable world-wide multicast infrastructure.

Napster and Its Legacy While IP multicast was being studied and improved by academics world-wide, the Internet was undergoing an enormous development and many things were evolving at a fast pace. The end of the ban to commercial activities marked the death of the 'elitist' Internet (mostly used by students and researchers) and opened it to the 'common people'. To the dismay of the purists of yore, e-mail spam started to fill the once sacred mail boxes and the newly-born World Wide Web rapidly became a shopping mall polluted by animated GIFs, HTML <BLINK> tags, annoying Javascript, and ubiquitous (and slow!) Java applets and ActiveX controls. To the enjoyment of the new dwellers, Internet was gaining appeal as a lot of opportunities for fun and profit were rapidly unfolding.

Among the plethora of software and services available on the Internet, a low-profile website - called Napster - began offering in June 1999 a program to download and share music files. The interesting feature of this program was that the music was downloaded directly from the other users: this provided fast access to the music files (as they were not located on an overloaded central server) and allowed a user community to develop improving the ability to find a large body of rare content.

Technically, Napster was a centralized search index where all users published the availability of their music for download and where they could search for new content. The results of a successful search on the Napster server allowed the Napster client to connect and directly download the music from another client. Nonetheless, Napster is regarded by many observers, along to SETI@Home[4], as the initiator of the so-called "peer-to-peer revolution", as it contributed to de-

[4]SETI@Home (http://setiathome.berkeley.edu/), a project launched in May 1999, is another early example of distributed application. The SETI@Home clients harnessed the power of idle CPUs to search for radio signals from extra-terrestrial sources.

velop the awareness about the power of distributed applications running on a huge number of computers around the world.

While Napster was already in trouble due to several pending lawsuits (it was eventually shut down in late 2002), Justin Frankel (Nullsoft) released in March 2000 an early prototype of Gnutella, a generic file-sharing program that overcame a major weakness of Napster - the central indexing server - by replacing it with a distributed search function. The Gnutella protocol and software immediately started to be extended and developed collaboratively by enthusiast programmers around the world, with the eventual creation of the Gnutella Development Forum. By then, several other peer-to-peer file-sharing protocols had begun to emerge, such as KaZaA, and new applications with a peer-to-peer architecture had appeared, such as Freenet [32] (anonymous and censorship-resistant information distribution), BitTorrent [33] (large-scale bulk content distribution), Skype (Internet telephony), PeerCast [14] (media streaming), etc.

The Peer-to-Peer (r)Evolution Distributed systems and algorithms, however, are nothing new, neither in theory nor in practice. The study of distributed algorithms was introduced a long time before, during the early 1960's, as the issues of concurrent program execution on multiple local processors and/or distant computers began to arise. The first practical instance of a distributed system on the Internet (then still known as Arpanet) was arguably the SMTP mail delivery system (RFC 821, 1982): its root ideas can be traced back to RFC 524 (1973), which denounced the limits of the earliest FTP-based mail protocols. The key concept of the protocol is the distributed message relaying performed by local servers toward the final destination. Another early example was the USENET protocol, in which the news servers exchange the relevant newsgroup data - as requested by their users - in a distributed way without relying on a central entity.

2.1.3 On Current Peer-to-Peer Networks

The main differences between the older 'distributed' protocols and the new peer-to-peer (P2P) applications can be especially seen in the following aspects:

- the functionality they offer: SMTP and USENET are limited to a store-and-forward data distribution model. Current P2P applications span a larger range of possible applications, from distributed search [106][73], to store-and-forward / bulk data distribution [33][32], to live data distribution [27], to networked virtual environments (NVEs) [63], etc.
- the interest of end-users in running the application and (consequently) the potential scale of their deployment: end-users are seldom interested in running SMTP/news servers, which require a relatively high availability and give relatively little advantage over using the POP/IMAP/NNTP servers provided by their ISP (or by a third party). For this reason, in general, each Internet domain has no more than a couple of mail servers and a news server: this fact limits the number of entities that participate in the distributed system to the order of the 10^4-10^5 nodes. A successful P2P application can attract about one or two

orders of magnitude more simultaneous users (e.g. more than $4 \cdot 10^6$ for the KAD DHT used by the *e-donkey* file sharing application [105]).

- the sophisticated techniques used by P2P applications to avoid dependence on Internet infrastructure: while SMTP performs lookups thanks to its tight integration with the hierarchical DNS system (MX records), peer-to-peer systems usually rely on their own addressing space and routing system, exploiting the DNS just occasionally and for standard functions such as IP address lookup and reverse-lookup.

We believe that the true revolutionary aspect of P2P lies in the practical impact that such systems have had on the mindset of both computer science researchers and Internet users.

For the researchers, it was a confirmation about the expected effectiveness of distributed algorithms. The success of early P2P systems motivated new research about scalability issues with an increased awareness of real-world constraints. We have witnessed during the last five years a definite trend of convergence between theoretical (graph theory, game theory) and system research (performance analysis, optimization) that has led to the development of many delicate and interesting trade-offs between the two traditional camps of computer science.

For the users, apart from the immediate advantages brought by the use of P2P systems ("free" music, "free" movies, "free" software), we can argue that these applications contributed a lot to the development of some sort of *user awareness* about their new role in the "information economy" and of higher *user expectations* from the "Internet experience". These aspects have recently evolved into the "Web 2.0" fad, which has more to do with the way users interact with Internet-based applications than with the technology itself. However, it cannot be denied that the increased focus on user interactions and user-contributed data has opened many interesting technical developments, which would have hardly been possible without the widespread success of P2P networks.

General Principles of P2P Networks A peer-to-peer network is a system composed two or more *peers* (or *nodes*) that exchange information over a computer network. Peers are *software processes* that "speak and understand" a common protocol, and may run on distinct pieces of networked hardware.

A peer-to-peer system does not need to rely on a fixed infrastructure, as long as its primary functionality is concerned. This definition concedes that, for the correct operation of the system, it may be sometimes necessary to rely on a centralized mechanism to overcome few specific problems which would be difficult to address in a purely decentralized manner. A typical difficult issue is the *node bootstrap problem*, which is common to all peer-to-peer systems because of their very definition: since there is no fixed infrastructure that supports the system, and with no initial knowledge about the current members of the established network, how can a new peer ever join the system? The solution adopted by most peer-to-peer systems is to rely on some external infrastructure or on a known reliable source of information, which allows to collect up-to-date bootstrap information. For this reason, a distributed system's solution to the bootstrap problem does not usually count as to determine whether it is peer-to-peer or not.

Peers operate autonomously based on their local knowledge. One of the main concerns about distributed systems is scalability, that is their ability to reach large sizes without suffering from excessive overhead. In P2P applications, overhead mainly depends on the amount of computational resources and information that is required by each node to correctly execute its algorithms: thus, scalability requires both a small number of connections to other peers and the access to a *partial view* on the whole system. The main challenge of distributed algorithms is providing the same features as a centralized algorithm, while just relying on a subset of the global knowledge which is *neither guaranteed to be consistent nor up-to-date*.

Peers are functionally equivalent, at least potentially. A (somewhat weaker) requirement for a distributed system to be called peer-to-peer involves a connotation of equality between the entities that participate in it. This does not mean that all the entities have to provide the same function or cover the same role *at the same time*: it rather prescribes that every peer, under appropriate environmental or internal conditions, could take up any of the roles or functions. For example, a P2P system might well adopt a hierarchical design (e.g. the 2-tier Gnutella 0.6 network, in which nodes are either *peers* or *super-peers*) where nodes behave in different ways depending on their current role in the system, as long as each node is allowed to assume either role.

An Interim Solution, here to Stay Whether a true revolution or a simple evolution of old concepts in the new Internet landscape, P2P networks are today very successful, as they definitely offer practical advantages in a number of applications. The traffic generated by P2P has become one of the strongest components of network traffic at the ISP level (ISPs and other sources[5] cite BitTorrent and other P2P traffic taking up about 80% of the total bandwidth near the backbone) and their popularity keeps increasing: despite the dissuasive tactics employed by media, recording companies, and film studios, despite technical interventions by ISPs aimed to selectively degrade the performance of P2P applications, and despite the adoption by technically ignorant lawmakers of draconian laws introducing bans on a vague "P2P technology" and punishing copyright infringement as a felony.

The success of P2P systems can be explained with the ease of their deployment, which usually requires nothing more than the installation of a single piece of software by the end-users. This requirement has been so light that, during the last decade, the evolution of the Internet and P2P networks has been largely symbiotic: on the one hand, the rising popularity of Internet has contributed to the growth of these systems, as the number of new users with Internet access has grown and still grows at a fast pace; on the other hand, the functionality offered by the P2P programs has fueled the growth of the Internet user base, as the awareness of the advantages of having an Internet access at home ("free" music, etc.) started spreading, especially among the youngest.

What in our opinion suggests that the P2P architecture is going to be a permanent part of the

[5]A 2004 CacheLogic study (no longer available) was often cited in the press. A copy can be found at http://web.archive.org/web/20041114005733/http://www.cachelogic.com/research/slide12.php . These measurements largely match the results reported by CAIDA in [61].

future Internet landscape are the impressive achievements of several families of distributed algorithms. The study of distributed hash-tables (DHTs) has led to the development of practical systems (like KAD, an implementation of Kademlia [73]) that can withstand reasonable churn and perform key-based search with $O(logN)$ or better latency [52]. BitTorrent [33] has proven a simple and powerful method for distributing large files to large user populations with little resources at the original publisher. The study of random graphs has made possible great advances in probabilistic data diffusion protocols, e.g. gossip protocols like LPBcast [41], SCAMP [47], and Swaplinks [115], and has also given birth to efficient unstructured keyword-matching search, as the recent Bubblestorm [109]. Finally, we must acknowledge that the recent versions of the Gnutella 0.6 protocol, which is based on partial flooding over a 2-tier overlay, keep operating in a satisfactory way well beyond the initial expectations.

Most important, all these achievements were possible without modifying the existing Internet infrastructure, without the need or even the expectation of back/forward compatibility, without a complex public standardization process, without the backing from big companies and industrial interest groups. Compared to the hurdles required to introduce new functionality in the core of the network (IP multicast, IPv6, etc.), operating at the application layer grants P2P systems a much faster development and deployment cycle, since it allows an immediate experimental validation on a small and medium scale, that can be followed by a seamless transition to the actual deployment.

This aspect makes P2P an interesting approach for algorithmic research and evaluation: as “experiments” can be performed in realistic environments and using large numbers of networked machines, it becomes relatively easy to obtain results early-on, which allow to test the working hypotheses and help with a successful theoretical modeling.

2.2 Basics and Requirements of Media Streaming

The concept of *media broadcasting* is a familiar one: distributing the same video/audio data intended for *immediate consumption* from a *single source* to a *large audience*. The best-known examples of this transmission model in our everyday life are the television and the radio networks. As commodity electronics available today are way more powerful and flexible than in the past, the user of a traditional broadcast system can expect a variety of options for the consumption of the received media that goes far beyond the immediate consumption *(live)* model allowed by television and the *store-and-playback* model allowed by VHS systems. Thanks to the presence of large quantities of shared-access data storage and CPU power, personal video recorders (PVRs) introduced new consumption models such as *time-shifting*, i.e. the possibility to seamlessly pause, resume, and rewind/fast-forward a live event while it is broadcast on television. The word *streaming* has been coined quite recently[6] to describe the analogous act of broadcasting multimedia data over a computer network, such as the Internet.

[6]The first uses of the term *streaming* appear to date back to around 1995, when RealNetworks began to market its RealPlayer program. The probable etymology of the word is an old logging term: it derives from the common practice by lumberjacks of using rivers and streams to transport the tree logs cut into pieces to the processing facility.

Transmission of data being the primary purpose of a computer network, there is nothing particular about transmitting media streams - rather than, say, text or pictures - over the Internet. The network is supposed to treat all kinds of data in an uniform way[7] as they actually are nothing more than mere strings of bits (cut and packed into datagrams) that have to be forwarded to their final destination. Contrary to the transmission of information by radio waves, which is subject to administrative oversight in terms of physical parameters (power, frequency spectrum, type of modulation, signal encoding) and thus strictly limited in terms of the maximum rate it can achieve, streaming supports a wide range of arbitrary *transport formats* and *coding formats*. Thus, the most important and defining parameter for a *media stream* is the data rate at which it is encoded, also called the *stream bitrate*, which corresponds to the amount of data per second required to reproduce the original media.

Internet-based streaming offers all the options that are expected from traditional broadcast systems. In addition, they can offer a much wider choice of media quality, live channels and stored contents. Compared to other applications that perform data distribution, the peculiarity of streaming systems lies in the time-sensitivity of the data they handle, which in turn derives from the user expectation of continuity, quality, and timeliness in the reception and reproduction of the media stream.

2.2.1 Time Matters!

The expression “streaming systems” encompasses a whole family of applications that deal with the distribution of multimedia content: the distinction among the various applications stems from the *different amount of timeliness* that is expected by their users as they receive and reproduce the media stream. We enumerate these systems in ascending expected timeliness order, that is *from the least to the most time-critical* (Table 9.1).

Bulk Media Distribution In bulk distribution, the media is actually treated as a regular file. Files have a *finite size and a content that cannot change over time*; they are considered opaque data objects and there is no special order (sequential, priority-based, etc.) following which their data have to be retrieved. We may assimilate the requirements of this streaming application to those found in file sharing or in bulk data distribution applications. Downloading a video via FTP, participating in a BitTorrent session to get a recent movie, looking up and retrieving a media file on E-donkey, Gnutella or KaZaA: all these actions can be considered as examples of bulk video distribution.

Video on Demand (VoD) Video on Demand (VoD) describes the distribution of recorded media data (e.g. video files) in a way that makes possible their consumption *while they are being*

[7]We are simplifying here for the sake of clarity. The adoption of Diffserv or traffic shaping techniques can introduce differences of treatment between IP datagrams at network layer – which are usually based on applicative requirements (QoS) or economical concerns.

Name	Example App.	Timeliness from Media Generation	Timeliness of Consumption
Bulk	BitTorrent, E-donkey	None (stored file)	Play media after complete retrieval
VoD	Joost, Youtube	None (stored file)	Low - Play during reception (1 min)
Live	Peercast, PPLive	High (10 $\sim$ 30 s)	High - Play as soon as possible (10s)
Interactive	Conferencing, Skype	Highest ($\sim$ 100 ms)	Highest - Play immediately

Table 2.1: Outlook on Video Streaming Applications

retrieved. The media data are again *static*, just like in the case of bulk data distribution: stored files of finite and known size. VoD introduces a first loose constraint on the way the data must be delivered to the player application: the user of a VoD system expects to be able to begin the reproduction of the content *shortly* after he initiates the data retrieval. The consequence of this constraint is that data must be made available to the player in sequential order to be reproduced: if the required data are not available the playout will be disrupted (*starvation*).

In VoD, while data from the whole media file *can* be retrieved out of order, the data that are needed for playout have to be recovered with higher priority. Also, the *buffering delay*, i.e. the time between the beginning of data retrieval and the beginning of media playout, must be chosen with care: it must be large enough to compensate for *all* the temporary starvations that will be encountered during the entire streaming process, but short enough to be acceptable to the user. The smoothness of the data playout process depends largely on the choice of both playout delay and data distribution policy, and is not directly related to the instantaneous speed of data reception.

The simplest VoD application can be conceived as a client-server protocol, such as FTP or HTTP, which provides the media file to a player application at a constant rate: if the average download speed is equal or higher than the stream bitrate, the playout may start few instants after the download is initiated. If the download speed is lower than the stream rate, the stream length comes into play to calculate the optimal initial delay required to avoid starvation for the entire (expected) playout duration. More advanced VoD schemes can download data in different parts of the file at the same time and from different sources: in these cases, the data that are nearest to the playout deadline are given priority over the others (e.g. as in BiToS [116] or RedCarpet [10]).

Live Media Streaming Live streaming is the first application in the data distribution family that deals with *practically infinite* streams of data and introduces a constraint on the maximum tolerable data reception delay. Live streams are different from the recorded VoD streams because:

- The current stream content is being distributed by the video source *only right now*, and will cease to be available in a short time
- The total duration of a stream is *not known a priori*: the users *join the system in the middle* of a streaming session, and *leave before the end*. The time spent by a user in the system

can be considered negligible when compared to the duration of the whole stream (hence the definition of practical infinity)

- The receivers of a stream are interested in reproducing it *with a reasonable delay*, as the interest for its data is highly volatile.

In addition to the two VoD requirements (sequential ordering of data to player, setting playout to avoid starvation), the requirement of *reasonable reception timeliness* characterizes the live streaming application. Because of the inherent properties of live streams, new data are constantly being generated by the source and need to be played, while older data quickly lose their interest as newer data appear: live streaming thus introduces the concept of **playout delay**, defined as *the delay between the generation of the media by the source and its reception by the viewer*. Contrary to VoD streaming, where users are largely independent in their behavior, the reception of a live stream becomes a *loosely synchronous* operation, since all the users are interested in roughly the same segment of the stream data at the same time.

Interactive Video Interactive video is at the most time-sensitive end of the spectrum of media distribution applications. Its requirements are very similar to those of live streaming application, as detailed above. The main additional challenge of interactive video is brought by the low tolerance to delay, which is typical of interactive applications: for a system to be qualified as interactive, the users should expect a very fast response between their actions and the reaction by the system, with a latency not exceeding the 100-150 ms range [26].

Conferencing applications are a typical example of interactive media: as every listener can intervene at any moment, the presence of larger delays may lead to contention between speakers and degrade the user experience. Because of the hard constraints on reception delay, interactive video requires a high synchronization between the users, is very sensitive to packet loss and network congestion, and makes it extremely difficult to implement distribution techniques that modify the sequential ordering of the data.

2.2.2 Media Quality Also Matters

The purpose of media streaming applications is to distribute time-sensitive data to several users at the same time over a computer network. Several common problems can however hinder the correct and on-time delivery of the media data, for instance: packet loss in the network, possibly caused by corruption, congestion (wired) or fading/shadowing/interference (wireless); excessive packet delay due to network congestion and queue build-up at intermediate routers; “route flapping” due to temporary instability of the inter-domain routing tables. The temporal granularity of these transient events can produce delay oscillations that can reach several hundreds of *milliseconds*. When the application imposes a media playout delay in this order of magnitude (e.g. in real-time interactive client-server applications), packet losses are not recoverable with an Automatic Repeat-reQuest (ARQ) scheme, and even an excessive delay in the reception of a packet makes it no longer useful for playout - with the same net effect of a loss.

This rather extreme example illustrates the fundamental trade-off between *media quality* and *reception timeliness* which is common to all streaming applications: the concept of *data loss* and thus of *quality degradation* is mostly determined by the timing constraints that can be tolerated by the application. When the timing constraints are strict as in the interactive case above, it is very difficult to *recover* from data losses, so it becomes reasonable to try to *prevent* them by using *forward error correction* (FEC) or other redundant coding techniques. On the other end of the spectrum, in the case of bulk video distribution, there is no reason why the media received without any timing constraint should be incomplete. In the middle of the spectrum, for VoD and live streaming applications, there is a substantial margin of tolerance to delay that can be exploited by the designer of streaming systems: by choosing appropriate timing constraints and loss recovery and prevention strategies, it becomes possible to elaborate innovative system designs that offer both acceptable delay and adequate quality.

Startup Delay as an Aspect of Media Quality For a long time, minimizing the playout delay was implicitly assumed to be the critical challenge of P2P live streaming [14][29]. While this is true for *interactive* applications, such as conferencing, the distribution of live streaming media allows much looser bounds on the playout delay, which is nowhere near to the interactivity threshold. Instead, an aspect of live streaming that is more likely to influence the perception of the user is the *time required by the application to start displaying the media stream*. This delay, known as *click-to-play delay* (C2P) or *startup delay*, is the time lapsed between the instant a user launches the streaming program (*click*) and the instant at which the playout begins (*play*). The startup delay is far more important in practical terms than the playout delay: while the user has no means of evaluating ’how fresh’ is the data he is receiving (unless he can relate it to an absolute time reference or to external events), he can surely tell how long he has been waiting for the player to start to produce its output.

2.2.3 Application Design: An Open Debate

To perform data dissemination in a network without support for native multicast functionality (*one-to-many*), it becomes necessary to build several application-layer “channels” between pairs of nodes (*end-to-end*). A centralized approach to dissemination involves having one node, the central sever, maintain a connection with every other node in the system. This scheme implies a *linear increase* with respect to the number of nodes in the system of the upload bandwidth required at the server that provides the streaming service. The server-based approach also introduces a hard limit to the size of the streaming (no more users can be served than the server’s upload capacity allows) and the problem of adequately provisioning the server infrastructure according to the expected audience size.

Distributed approaches can successfully reduce the total upload capacity needed at a single node. The use of a distributed architecture not only lowers the costs for the publisher, but it also confers self-scalability to the system: as long as each user “gives back” to the system at least as much as it requires to be served, the size of the streaming audience can in theory increase without bounds.

However, so far no consensus has been reached about the exact requirements of a practical Internet streaming application, and the debate about how to design P2P live streaming applications for large scale deployment is far from being settled. While many authors agree about most of the challenges, there is still no widespread agreement on what architectural design offers the best performance for a large range of applicative scenarios. The debate is still open on fundamental points as:

1. **Basic features of a live streaming system**. Traditionally, low latency has been considered the primary property needed for a viable streaming infrastructure. The first practical distributed applications [14], however, showed that the instability of software nodes (e.g. compared to routers) had a largely negative impact on the streaming performance, as the failure of single peers had cascading consequences over the rest of the system. Resilience to churn and failures have thus been recognized as a further requirement: the acceptable trade-off between latency and robustness for the different applications is still an open topic of discussion.

2. **Efficiency of a streaming system**. In the early work about P2P live streaming (better known back then as "application-layer multicast") the term of comparison for efficiency was native multicast: metrics as link stress or stretch were routinely used to evaluate the latency/data replication overheads of application-layer trees with respect to network-layer IP multicast trees. The appearance of mesh-based systems has rendered such comparisons more difficult, as data do not keep flowing over the same path even while the node membership remains stable. Research is still ongoing on optimal theoretical schemes for chunk scheduling and bandwidth allocation [103][34][72].

3. **Robustness under real-world scenario**s. How to characterize the behavior of a live streaming system under churn? How to fairly compare the control overhead of mesh-based systems, which is almost constant with the churn rate, against the overhead for tree repair, which instead is very sensitive to churn? What is the critical churn threshold at which each system stops operating correctly? How does content degradation happen in each system once this threshold is reached?

4. **Optimality of streaming under capacity constraints**. How is optimality defined in networks where the node upload capacity is not uniformly allocated across the system and/or where the resources are globally scarce? How will P2P live streaming systems perform under an arbitrary bandwidth distribution? How will a P2P system behave in a non-cooperative environment?

5. **Topological awareness**. How can the awareness of a system to the underlying network conditions be evaluated? How can its latency overhead be improved compared to native multicast?

2.3 P2P Live Media Streaming Applications

Early P2P Designs Three landmark works can be seen as the most influential precursors of more recent live streaming systems. The first [99] describes **chaining**, a simple technique to allow the distributed replication of streaming media to small-to-medium audiences. Data are replicated by each user to the next in a linear chain: while the system offers a way to achieve self-scalability for the first time, its delay performance is a poor $O(N)$; moreover, node arrivals and departures have a strong negative effect on data reception, as the entire chain is broken at some point.

Yoid [44] is a proposal by Paul Francis for a generalized applicative multicast infrastructure. The structure of Yoid is twofold: it consists of a mesh of connections between the nodes, which is optimized for robustness to avoid network partitions and to ensure member reachability, and of a tree, built to offer optimal performance in terms of delay and bandwidth, that actually conveys the content: these two topologies are largely independent as they have different goals. Yoid explores a peculiar *mesh+tree* approach, where a primary concern is to avoid loops in the tree: an evolution of this design has resulted in the recent development of Chunkyspread [113].

Narada [30] is a system designed to support multipoint-to-multipoint communication such as small-scale video conferencing. Narada organizes the nodes in a mesh, over which a distance-vector routing algorithm allowed to build spanning trees that support the actual data distribution. The mesh overlay is updated over time by establishing new connections to random peers: the decision whether to keep the new neighbor is made on the basis of the improvement in the average cost to reach all the other nodes. The use of pairwise latency between the nodes as the cost metric allows to optimize the trees for minimum delay. The improvements to the initial design introduced in [29] add the link bandwidth as a second optimization criterion for tree construction. Narada suffers from scalability issues, as every peer has to keep state for each one of the N other peers in the network.

Two-Tiered Streaming Systems Several designs of large-scale live streaming systems with a distributed component but based on infrastructure were introduced in the same years [59][25][24]. These proposals were made with a service-oriented scenario in mind, in which an organization such as a telephone operator or an ISP would provide video broadcasts to *its customers* over the Internet (or rather, on their internal network): as the content providers would operate both the infrastructure and the streaming application, they would be able to retain control over the system and possibly offer a guaranteed level of service quality.

The common feature of these systems is the use of a **two-tier model**, comprised of a *peer-to-peer core*, where all the nodes are server-class machines operated by the service provider, and the *clients*, which receive service by the core network without providing any resource. The core network can be connected by either unicast connections or locally-scoped multicast. Users establish connections to the servers under the supervision of a load-balancing mechanism that can be either internal or external. Data are transferred over single connections between each client and its assigned serving node.

Overcast [59] builds trees between the servers that aim to maximize the bandwidth availability from the source to the leaves. This algorithm also allows the shape of the tree to loosely follow the underlying network topology. Load balancing and user admission are performed in a centralized way by the root node of each tree, but these functions could possibly be distributed to improve the global robustness against failures. Scattercast [24] is a system conceptually very similar to Overcast, with an additional feature: the nodes in the core network, called SCX (ScatterCast proXies), can adapt the quality of the content they deliver to the download bandwidth and to other requirements (such as particular real-time constraints) of the nodes they serve. RMX [25] adds features on top of the Scattercast framework, such as the use of locally-scoped multicast by the proxies to serve clients in a same location, and is meant better support bandwidth heterogeneity and data losses.

2.3.1 Current P2P Live Streaming Systems

Single-Tree Overlays The subsequent approaches to P2P live streaming [12][110] inherit much from the earlier research on application-level overlays mentioned above. A single-tier tree overlay was chosen as the simplest topology to convey data to large user populations: the good scaling properties of trees and the relative similarity to native multicast motivated the earliest approaches, such as SpreadIt [14]. To improve scaling and resilience, NICE [12] later proposed a hierarchical, cluster-based single-tree overlay, which allows to optimize the average delay through appropriate node management policies and cluster-head selection criteria. ZIGZAG [110] further improves on this design by adding redundancy to the cluster management mechanisms to better cope with node churn.

The fundamental shortcoming of all tree-based systems is due to the limitations imposed by the tree structure: single trees artificially limit the available service capacity as the leaf nodes, which make more than half of the population, are prevented from contributing bandwidth to the system. Moreover, in a real-world context, there are no guarantees about the resources brought to the system by nodes involved in a streaming session. Nodes can be limited in their contribution by the access technology they use (e.g. slow ADSL uplink), by other applications that compete on their upload bandwidth (e.g. other file-sharing applications), and/or by IP connectivity issues - such as the use of network address translators (NAT) or packet shapers at the ISP. Other limitations may be voluntarily introduced by the user, e.g. application-level bandwidth limiting.

Multiple-Tree Systems Multiple-tree overlays were proposed as a solution to the inefficient use of upload capacity by single trees. By encoding the stream as several independent *MDC stripes* [50] and distributing them over different trees, these systems can exploit the upload capacity of leaf nodes and spread the load uniformly across the whole population. CoopNet [80] first introduced this approach: while still relying on a resourceful central server, the client nodes could cooperate to the content distribution in case of high server load. Splitstream [21] is the first P2P system that uses interior-node-disjoint trees. Every peer is an interior node in no more than one tree: this mitigates the incidence of the data losses due to node failures, as isolated failures result in the interruption of at most one stripe, which can be masked by the encoding method. On the other hand, the control overhead is higher than in the single-tree case: in general, multiple-tree systems must rely on an underlying DHT substrate for tree-building and maintenance purposes. Further research has however shown the limits of a DHT-based overlay multicast protocol: in [16] the authors observe the shortcomings of Splitstream under churn and in presence of upload heterogeneity, as its trees become much deeper than the expected theoretical depth and rely on non-DHT links for a large number of connections. These phenomena noticeably degrade the system performance and scalability: [16] argues that DHT-based schemes cannot easily support heterogeneity and churn, because of the fundamental mismatch between node constraints (unknown a priori) and randomly-assigned DHT identifiers.

Splitstream had a major influence on the subsequent development of P2P live streaming: a large number of systems quickly adopted the multiple-tree architecture, developing original tree topologies in order to improve the properties of the tree-building algorithms [46][97]. More recently, Chunkyspread [113] has removed the requirement for a structured substrate such as a DHT by adopting a clever non-hierarchical approach to tree-building. Using a gossip protocol, nodes exchange the list of the data stripes they currently receive along with a compact Bloom filter representation of the list of their ancestors for each stripe. Bloom filters are used to constrain peer selection and assure that the resulting stripe distribution paths will be free of loops. Peer selection is based on the *load* advertised by the neighbors and on their *latency* in receiving specific stripes. The unstructured architecture of Chunkyspread allows the use of other policies for peer selection, such as tit-for-tat: however, preliminary results [112] show a decreased performance when TFT-based approaches are used.

Data-Driven Systems Mesh-based designs aim to reduce the structural constraints of live media streaming systems. These systems break up the stream into a series of **data chunks**: the chunks are generated by the source, which then transmits them to a small number of nodes. After that, the nodes must autonomously exchange the chunks and retrieve a complete and ordered sequence before the expected play-out deadline. Bullet [65] is an early approach that combines a single-tree and a mesh: the tree is used to convey both data chunks and control information, while the mesh is created independently by the nodes based on the control information and is used to exchange the bulk of the data among peers that are far away in the tree hierarchy. An advantage of this scheme is that the control protocol running on the tree can have a very low complexity and bandwidth overhead. On the other hand, no mechanism to encourage bandwidth contribution was implemented in the system.

The emergence of unstructured mesh-based P2P live streaming systems begins in 2004: some examples are Coolstreaming, Chainsaw [81], PULSE [85], GridMedia [121] and, later, PRIME [70]. Chainsaw [81] is a proof-of-concept example of a simple mesh-only system that uses randomized peer- and chunk-selection algorithms. However, it has not been tested on scenarios with heterogeneous bandwidth. Coolstreaming/DONet [122] introduces a smarter chunk scheduling algorithm that takes into account the expected play-out time of the individual chunks, gives higher priority to the locally-rarest chunks, and distributes the chunks based on an estimate of the available capacity at the sender nodes. GridMedia [121] uses a combination of *pull and push* to supposedly improve the efficiency of chunk distribution. PRIME [70] proposes a peculiar mesh-based design which is actually more similar to a multiple-tree system: several separate tree structures are present over which different data is distributed, and mesh-based “swarming” is achieved by making the leaves of each tree serve data toward random members of the other trees.

In presence of upload heterogeneity or non-cooperative environments, however, chunk scheduling and peer selection are subject to additional constraints due to the uneven distribution of node upload capacity. While the optimal node placement can be achieved by having peers with higher upload nearer to the source (as demonstrated in [103], Lemma 1), information on the upload at the other nodes is rarely available (or, if available, not very reliable). Distributed algorithms that solve these issues in non-cooperative scenarios are still a topic of active research.

The Advent of Practical P2P Live Streaming Systems Meanwhile, the first working prototypes of practical live streaming applications started to emerge. Peercast [36] was released in 2002, and gradually attracted a small following of users and broadcasters of (mainly) radio channels. End System Multicast [27] was the first large-scale video distribution system based on a single-tree multicast infrastructure to have been deployed and for which measurements have been collected. The authors of ESM preferred to use well-understood technologies (a traditional single-rooted overlay tree, standard single-description coding for the media streams) rather than implementing more convoluted architectures, and were deeply concerned by real-world connectivity problems (e.g., firewalls and NAT). The main goals of ESM were to show that application-layer overlay multicast over the Internet was already feasible at the time, to highlight the issues and limitations of the basic single-tree system architecture, and to gain a first practical experience to drive future research on the subject. While ESM does not introduce theoretical models nor proposes original distributed algorithms, the *lessons learned* section of [27] contains very helpful advice about creating distributed applications and describes well the main problems that have to be considered in the design of practical overlays for live streaming. Thanks to its simple design, ESM has been used as a foundation in subsequent research projects: for instance, the ESM software was recently adapted to support multiple trees and collaborative incentive schemes [107].

Later, Coolstreaming [122] was the first practical P2P live streaming system to propose a completely unstructured, mesh-based design. Together with Chainsaw [81], it provided the earliest insights into the feasibility and advantages of this approach. A working prototype of the Coolstreaming software was released in 2004 and quickly became popular: more than 30,000 users were counted, with as much as 4000 simultaneous viewers. Coolstreaming introduced an interesting chunk scheduling algorithm (more complex than the random scheduling used in Chainsaw) that takes into account the individual chunk deadlines for play-out together with the estimated bandwidth intake from each of the serving neighbors. Also, every node performs an iterative long-term optimization of the mesh by replacing its least-contributing neighbor, in order to slowly adapt the overlay mesh to the variations in bandwidth availability inside the system.

Control Plane & Knowledge Mgmt.	Data Plane	
	Tree-Based	Mesh-Based
Implicit	NICE [12], ZIGZAG [110], ESM [27]	=
Structured	Splitstream [21] (Control: DHT)	Bullet [65] (Control: Tree)
Unstructured	Chunkyspread [112][113]	DONet [122], Chainsaw [81], PULSE

Table 2.2: Summary of Main Approaches to Live Streaming

Coolstreaming does not seem to adopt any specific measure to address the combined effects of upload bandwidth heterogeneity and high churn. The use of Virtual Coordinates [77] has later been proposed in [23] to improve the locality awareness of Coolstreaming overlays. Initial user reports about the Coolstreaming application seem to indicate that it suffers from a high play-out latency (up to several minutes), which is probably due to conservative data buffering policies.

After Coolstreaming was shut down in response to legal threats during 2005, many other applications with similar functionality quickly took its place: the more popular among them are today SOPCast, TVAnts, PPStream, and PPLive [102]. Recent measurement studies on PPLive [55] have revealed the astonishing success and impressive deployment status of this system, with a daily average of 400,000 users and average measured simultaneous audiences in the order of 100,000 viewers for the most popular individual channels.

2.3.2 Analysis of P2P Overlays for Media Distribution

An interesting aspect of live streaming is the large solution space that this application presents. The existing architectures adopt a number of original schemes of data distribution overlay. Table 2.2 summarizes the main design choices adopted by recent P2P live streaming systems: *mesh-based* or *tree-based*, based on the way data are exchanged by the nodes (**data plane**), and *structured* or *unstructured*, referring to the way control information and knowledge are propagated (**control plane** or **knowledge management**).

Knowledge Management: Implicit, Structured, Unstructured An important factor in the ability of a system to withstand transience is the way its overlay is built and maintained. We can observe three prevalent ways of maintaining the connectivity among the nodes:

- Implicit connectivity: each node knows only about those nodes to which it is directly connected. The earliest single-tree overlays were based on this scheme, which suffers heavily from node transience. To increase the robustness, information about additional nodes in the system has to be maintained: in [36] nodes contact their *grand-father* upon a parent failure, while in [27] all the ancestors on the path to the source and some more nodes chosen at random are known. We qualify their control plane as *implicit* because it is largely dependent on the geometric structure of these overlays.

- Structured membership management: the peers belong to an external infrastructure (DHT, tree, etc.) which either defines how the data connections can be established between the nodes [21] or provides a channel over which the updates about the buffer content of the nodes can be propagated.

- Unstructured membership management: the peers use a randomized gossip protocol to distribute and receive updates about their state and content of their buffers [122][81][85].

To perform live streaming a peer also needs, in addition to the basic information required to contact other peers, some form of up-to-date knowledge on their current data availability. The presence of *intrinsic instability* in the data retrieval process, as in data-driven systems, implies a reduced reliance on long-lived information about the rest of the system. If we consider the various random factors that can perturb the operation of a streaming system as sources of *external instability*, the presence of intrinsic instability can help to accommodate a certain level of externally-induced transience. Therefore, when the scenario targeted by the P2P application involves a high instability, the use of less-structured and dynamic systems will be more appropriate than a structured and static architecture.

Developing an *a priori* assessment of the level of churn a real-world application will have to sustain is challenging, as the user behavior has a large influence over this parameter. A strong interest has developed lately for measurement studies of widely-deployed streaming systems that aim to characterize the user behavior in real-world systems [9][55].

Data Plane: from Trees to Meshes The traditional approach to P2P live media streaming consists in creating a multicast infrastructure at the application layer, over which data will then flow [14][12][110]. **Tree** overlays, whose main property is the absence of loops, do not need any exchange of control information once the overlay has been built. Control messages are only required during the overlay construction phase and to repair the tree after node disconnections. During normal system operation, the information about which data will be received in the future is *implicitly conveyed by the placement of a node in the overlay*; also, since no loops are present and data are transmitted sequentially, there is no necessity for reconciliation mechanisms. We can thus consider these overlays as permanent channels, which are constantly used to distribute an ordered and steady data stream. The major drawback of single trees, as we mentioned above, is that they artificially limit the available service capacity, as the leaves cannot contribute any upload bandwidth to the system. Moreover, each internal node in a tree is a potential bandwidth bottleneck for the subtree it serves, and packet losses do *accumulate* while descending the tree. Finally, the maintenance, optimization, and recovery of failed overlay tree links can become a daunting task under heavy churn. Data losses that occur during the tree repair process can be partially masked and attenuated by buffering and recovery mechanisms, but - eventually - they end up severely disrupting the play-out quality.

Mesh-based (or **data-driven**) systems [122][81][55][85] do not explicitly define fixed data paths over which data will flow. The stream is no longer a continuous sequence of bits: it is now split into *chunks*, the basic units of data exchange, which are forwarded independently by each peer to its local neighbors. Mesh overlays are connected, directed graphs generated by the local connections between nodes, thus in general they do not satisfy any specific property. The only constraints that apply to meshes involve limitations on the inbound and outbound degree of each peer, which may vary from node to node. For this reason, a node of a data-driven system does not know anything a priori about the data it will receive from his neighbors: in particular, it is not able to predict the data rate it shall receive on each connection, nor can it foretell the order in which he will obtain the data chunks. Moreover, the connections that make up the mesh must be renegotiated periodically, to adapt the set of potential partners for data exchange to the actual availability of useful data throughout the mesh.

Tree or Mesh: a True Dichotomy? The tree and mesh approaches seem to be largely incompatible, as they deal with content distribution in antithetic ways:

- Trees require simple mechanisms to build and maintain paths that are loop-free. Assuming that losses can be either tolerated or recovered at the transport layer, data can flow over the paths without need of any signalization. The main issues with tree-based systems can be summed up as: building a good overlay using appropriate construction policies, which in general attempt to avoid to introduce slow nodes and/or bandwidth bottlenecks along the data paths, and protecting the integrity of the overlay and its geometric coherency against node failures and disconnections, which can become a big problem in case of high churn and node transiency in large overlays.
- Meshes are the global result of the local associations of nodes: they do not follow any global structural criteria. Frequent control exchanges are required to avoid the redundant duplication of the individual pieces of data (*chunks*). The main issues with data-driven systems are defining efficient algorithms for peer and chunk selection, and providing nodes with critical information about data availability at their neighbors in a scalable way.

The recent developments in the field have introduced several intermediate solutions, which - as seen from the tree perspective - give up a certain amount of determinism in the system structure to introduce a better support for network dynamics, or - as seen from the mesh point of view - introduce some structural constraints in exchange for a reduction of the control traffic. **Multiple-tree** systems like Splitstream can indeed be seen as an incarnation of the former trend, since the final result is a mesh based on geometrical rules (trees) and additional constraints (tree disjointness, etc.): the existence of several independent data paths improves the overall resilience to node failures and data loss. The latter trend is evident in the **mesh+tree** approach in Chunkyspread [113]: by tagging the data forwarded by each node in an appropriate way, loop-free paths can be dynamically created over the mesh which can be re-used for many subsequent pieces of data.

However, if we look closely, it becomes easy to realize that the “philosophical” incompatibility between trees (fixed structures, constant data paths) and meshes (no a priori organization, variable data paths) is only apparent: indeed, the fundamental task of an efficient streaming system is to transmit every individual piece of data to every node exactly once. This means that in both cases the data will follow a path which is an acyclic directed graph, i.e. a tree: this graph is either built explicitly and re-used over a long time frame (overlay trees) or built implicitly and constantly changed in the data-driven meshes. From the point of view of system performance, multiple-tree and mesh-based architectures can attain the same efficiency and scalability.

2.4 Conclusions

It is our opinion that the design space of P2P live streaming applications has been thoroughly explored. We also remark that the existing architectures cover the full range of possible combinations of structured and unstructured solutions in both, control and data planes (see Table 2.2). Another fundamental observation is that, whether using meshes or trees, the path taken by any single piece of data always ends up to be a tree. This intrinsic similarity renders multiple-tree-based and mesh-based system very similar from the point of view of both efficiency and scalability. The differences mainly reside in the granularity of the knowledge about the rest of the system and in the expectations about future data reception, rather than on the way the data distribution overlay operates.

We are convinced that knowledge and expectations, which describe the *flexibility* of a system, should be the primary object of a system designer's attention. Given a target deployment scenario and its expected churn, a reasoned choice between a structured or unstructured approach is relatively easy to formulate: when churn is low, structured overlays are more efficient, as they require a very small control overhead when repairs are sporadically needed. When churn is high, unstructured overlays guarantee higher resilience with a constant control overhead. For intermediate churn scenarios, both choices are equally viable, and the implementers can often adjust some parameters such as the frequency of control messages, the amount of information about other nodes, etc., to best adapt to the operating scenario.

Mesh-based systems give an intrinsic advantage from the point of view of topological optimization over structured tree-based systems, since they 'naturally' require a larger base amount of knowledge on the system. This advantage can however be reproduced in a tree-based context by introducing random gossiping and advertising a larger amount of internal state [27].

The main argument that - we believe - gives a major advantage to unstructured systems is their better suitability to non-cooperative environments, which directly translates into a better support in heterogeneous upload capacity distributions as found in most real scenarios. By using the received bandwidth to support some form of pairwise incentive, such as tit-for-tat, meshes can promote a form of intrinsic optimization and resource awareness that is solely based on local information and which is not possible in a tree-based scenario, unless an external mechanism is introduced to certify the contribution of each node to the system [89]. The most elaborate mesh-based systems, such as Coolstreaming [122] and Chunkyspread [113], all recognize this potentiality as an advantage over systems based on static data paths. One of the goals of PULSE is to explore a specific form of practical incentives that promises to support both, resource and locality optimizations, at the same time.

Chapter 3

The PULSE System

In this chapter[1], we describe the PULSE system. In Section 3.1 we introduce the goals of PULSE, how it was born and evolved over time, which intuitions guided its design, and what we believe are its original contributions.

Then, we present in full detail the inner workings of the PULSE system. We start by illustrating in Section 3.2 the notations and basic terminology we employ in the rest of this work. We proceed by describing the basic component of this distributed system, the PULSE node. Section 3.3 is a top-down overview of the different functional components of the node. Here we define the purpose of each component, its main data structures, and the way data are managed and updated internally. In Section 3.4, we cover the algorithms used by a PULSE node and enumerate the main parameters that affect its operation.

Section 3.5 contains a brief overview of the prototype of a PULSE node, which has been implemented to validate the algorithms and to test the behavior of the application in real-world settings.

3.1 Introducing PULSE

To introduce properly the main subject of this chapter - and focal point of this whole thesis - we will begin from the name of our system. A long-standing engineering tradition prescribes in fact that a project should be described by a self-explaining acronym, with bonus points for the semantic value and recursiveness of the acronym.

PULSE is actually an acronym that stands for *Peer-to-Peer Unstructured Live Streaming Experiment*. This denomination includes some of the fundamental design choices on which our system is based: the fact that it is peer-to-peer, i.e. does not require the availability of a centralized resource for its operation; the unstructured nature of the system, which does not rely on a fixed

[1]The contents of this chapter have been published in part as [85] and [87].

topology for the exchange of data and control information; and the fact that it was an experiment, as the viability of this kind of approach was not clear from the beginning. The acronym was born during a hot day of July 2004, while the author was still drafting the early specifications of what was - until then - an unnamed live streaming system capable to work in realistic network environments.

3.1.1 The PULSE Manifesto

The primary purpose of the PULSE system, which was devised in the beginnings but still holds true today, is allowing an Internet host - *any host* - to either act as a distributor of its own live media stream or as a receiver of a live stream from any other node. The technical challenges to this applicative scenario are:

- *Any host* means no guarantees on the stability of the peer. There are several reasons that urge us to consider peers as unstable and to design a system that tolerates node failures: Internet hosts are in general unreliable, as they may crash or go offline because of software glitches or user activity; also, a peer is a software process that can be activated and stopped by the user at any moment; finally, the users of a live streaming system are expected to interfere with the application in an unpredictable fashion, for example by 'switching channel' and associating to another streaming session. We will therefore assume that *the expected lifetime of a peer in the system is very short, in the order of hundreds of seconds*.
- *Any host* means no guarantees on the bandwidth resources of the nodes taking part in a streaming session, including the source itself. We do not want to restrict ourselves to largely optimistic scenarios, where all the nodes have sufficient resources to receive the stream and to replicate it at least once: currently, while the downlink of commercial ADSL Internet access is now sufficient to receive a TV-quality video stream (at 500 Kbps and above), the uplink capacity is often much lower, typically less than 500 Kbps. While the connection speeds, both uplink and downlink, have been steadily increasing in the last decade and will probably continue on the same trend, we also remark that the video quality and the bitrate it requires has kept growing [57]. Therefore, *in a medium-term perspective, we still believe that the bottleneck will continue to be located at the uplink for the majority of the nodes*.
- *Any host* means that there is no central access control facility that can verify the performance of a node and decide whether to allow or deny it access to the system. Every node is then allowed to enter enter the system, but the system itself has to cope with the balancing and allocation of the available resources. The system should then be expected to *work in a best-effort mode, with a graceful performance degradation in case of generalized lack of resources*.

This is the environment we are designing for: a strongly dynamic environment, with nodes joining and leaving continuously (*churn*), with peaks of arrivals (*flash crowds*) and departures, where nodes have an asymmetrical access link, where the upload bandwidth is not uniform but largely spread (and often inferior to the stream rate), where nodes are considered non-cooperative by default. An additional 'extrinsic' constraint that we want to introduce is the fact that the algorithms and code for our system should be made public and be easily modifiable by anyone, e.g. like the BitTorrent software.

Our choice of this kind of scenario is supported by several previous studies:

1. A milestone analysis of the bandwidth availability in a Gnutella network [96], performed in 2002, showed that node uplink bandwidths were distributed following an approximate power law. While five years have passed, we believe that the results of this study are still valid as long as we factor in the increase of average connection speed. More recent studies confirm that bandwidth asymmetry and scarcity of resources are still a concern: for instance [27], despite a more limited scope in comparison to [96], offers a rather bleak outlook based on data from measurements of medium-scale live streaming sessions on the Internet.

2. Studies about the role of churn in distributed live streaming systems [14][104] estimate the churn rate and its impact on tree-based systems[2], concluding about the need of appropriate overlay construction techniques. In its insightful analysis of real traces from a number of large-scale live broadcasts, [104] suggests the need for overlays capable to withstand transient peer behavior, such as multiple-tree forests, and advises against predictive mechanisms based on past node lifetime.

3. Experiences from other applications suggest that cooperation should never be expected from the members of a large-scale distributed system. An example is the well-known seminal study by Adar and Huberman on free-loading on the Gnutella network [8], which showed the inherent lack of willingness of provide resources to the system, if no benefit is expected in return. Similarly, it may be risky to rely on peer-provided feedback about resource availability or past performances, as this information can be easily tampered with by the user if, by doing so, some kind of benefit can be expected[3].

3.1.2 Background

Distributing live data to a large audience requires short data paths that grow slowly with the number of nodes that have to be reached. Single trees are useful in this context, as their depth

[2]A serious oversight that was quite common in the earlier works was to assume that the behavioral traces obtained from any distributed application could be seamlessly exported to any other: for example, traces from large scale file-sharing systems such as Gnutella were used in [21] to describe typical membership of live streaming sessions. Later studies have fully recognized that the type of application and the way it interfaces with the user strongly influence the actual user behavior, and by consequence the distribution of node lifetimes.

[3]In this document, we do not intend to consider advanced form of malicious behavior, such as strategically tuning the bandwidth contributions or willingly disrupt the functionality of the system. These concerns are more pertinent to the field of security, which lies out of the scope of our discussion.

increases logarithmically with the population size - $O(log_d N)$, if the degree d of each node in the tree is constant and if the tree branches are balanced. However, the dynamical growth of a tree-based system heavily depends on the capacity distribution of the nodes in the system and rarely leads to balanced trees [20]: in the real world, it is not possible to expect a fixed per-node bandwidth contribution, as it depends both on upload limitations at each node and on its willingness to contribute. In practice, this approach usually leads to poor performance, as very few nodes can contribute a sufficient amount of excess resources to the system. Also, a degree $d > 1$ implies that a node is contributing more resources to the system than it receives, which makes little sense in an economical perspective[4]. Finally, a large share of upload capacity is wasted, as the leaves are (by definition) prevented from contributing.

In the hypothetical case where everybody contributes equally, the single tree degenerates into one or more *chains*, whose length is $O(N)$ - not suitable for the timely distribution of live data. To introduce timeliness in this context, the multiple-tree approach has been devised: split the content in k stripes, which are then distributed using k internal-node-disjoint trees of degree $d = k$. The internal-disjointness requirement can be tuned by appropriate tree-building algorithms so that the load on each node is not higher than what it would have been in a single tree with $d = 1$, while the path length scales now much better than before: $O(log_k N)$.

Again, the real world does not guarantee that each node will contribute equally to the system. It is a common occurrence that nodes will provide less upload bandwidth than what nominally required. Solutions to this issue include: preventing 'poor' nodes from joining the system [21] (access control), eliminating 'poor' nodes by means of a external cryptographic records [89] or complex tree+cluster/based organization where neighboring nodes reach consensus on whether to enforce punishing measures [46] (detection). Few multiple-tree based system accept the fact that peers could contribute less than the stream rate. As far as we know, when they do, they expect that 'poor' nodes will either refrain from obtaining more resources than what they contribute [107] or politely comply with the demands from richer peers that try to push them towards the bottom of the trees[97] (cooperation). The effectiveness of these approaches is not guaranteed in non-cooperative environments (users can actively interfere with the correct node behavior, e.g. by willingly disabling data upload) and when the software code is public and could be modified and then recompiled.

What we expect from an Internet application is to work in best-effort mode as long as resources are available, and to fail gently when resources are no longer sufficient, based on locally-available knowledge and without relying on explicit cooperation.

3.1.3 Fundamental Insights

The initial intuition that motivated our work on PULSE is that there should be a better way of organizing the members of a live streaming session than using trees. "If we accept to break up the

[4] Actually, in a game theoretical perspective, it makes no sense for a greedy player to provide *anything* to the system, as there is no way that its lack of contribution can be detected and retaliated against.

rigid tree structure - we thought - it would perhaps become possible to leverage local information to optimize the overlay in a way that was not possible with traditional tree-based designs".

We also understood that asymmetry in upload resources (and/or the lack of willingness to provide them) is a fundamental problem that has to be taken into account early on, while defining the way peers interact and choose their partners: a system where cooperation between users and resource availability are expected would hardly operate in an environment such as the Internet.

Finally, we considered churn as a natural consequence of a ***live*** application: users would be joining, leaving, switching from one streaming session to another, rapidly zapping through a number of channels. The reliance on static, stable overlays that have to be actively repaired in response to churn appeared to us as an inappropriate choice, much less effective than renouncing to the classic concept of overlay that (back then) was ubiquitous in the literature.

These intuitions suggested us to devote our efforts toward an unstructured, data-driven, incentive-based, dynamic system design.

Unstructured Systems - Resilience to churn Building tree overlays has several shortcomings. When the incoming nodes proceed hierarchically (from the root to the leaves) the source is involved as a starting point [36]: this fact limits the flexibility of the system, as repositioning a disconnected node means recontacting the source and traversing again the $O(\log N)$ tree layers to get to the bottom. The solution to this lack of flexibility requires an increase in the node knowledge, to avoid getting back to the source every time the tree requires maintenance: keeping the contact information for nearby nodes, such as the *grandfather* [14], or maintaining the entire list of ancestors [27] plus some random nodes, allows to somewhat mask the effect of transience on the playout quality. However, in presence of limited bandwidth resources, a disconnected node will rarely be able to reconnect on a higher layer, as the capacity of its ancestors is likely to be completely allocated: several hops will be required for a node to reach its final position in the tree. The use of structured overlays, such as DHTs [21], for tree construction and maintenance solves the *reconnection* issue, as the underlying overlay layer implements repair policies that act transparently, ensuring the connectivity to the system in spite of node transience. However, the streaming application loses the control on peer selection and cannot be aware of factors such as bandwidth availability and latency [16]. Unstructured systems can support very high amounts of churn without adding constraints to the streaming application. Moreover, the less structured is a system, the lower is the impact of external perturbations on its operation.

Unstructured Systems - Flexibility in node placement The advantages of an unstructured system lie in the fact that the streaming application has a much larger degree of freedom in the way it manages its connections: this freedom requires a broader amount of information on a larger subset of the node population, but if properly exploited it can grant to the system the capability of adapting itself to variable network conditions. An example of these advantages is provided by the recent Chunkyspread architecture [113], whose unstructured tree-building technique allows seamless switching between different positions in the same tree while keeping into account both the load of the nodes and the network latency.

Data-Driven Systems - Flexibility in resource allocation A data-driven approach imposes less constraints on the contribution of individual nodes. While in traditional tree-based systems each node would have to contribute an integer multiple of the stream bandwidth, and while in multiple-tree systems nodes are supposed to cooperate, providing at least the same bandwidth that they consume, in data-driven systems there is no fixed data rate that has to be maintained on each connection. The free allocation of the node upload capacity brings additional flexibility in the establishment of peer associations and in the scheduling of data transfers.

Data Driven Systems - Dynamic overlay optimization The freedom from structural constraints allows unstructured systems to support non-deterministic optimization techniques to recursively improve the overlay quality: this interesting option was already implicit in the earliest examples of unstructured data-driven systems, such as CoolStreaming/DONet [122]. The awareness to pairwise node latency and to the availability of upload capacity among the peers are important features that can help lower the playout delay and improve the efficiency of data distribution [27].

Incentive-based overlay optimization The live streaming problem has several features that suggest the effectiveness of incentives to stimulate cooperation: first, it involves a pool of users interested in roughly the same data, synchronizing the attention of all the receivers on a small segment of the stream; second, it requires that the users stay in the system as long as they wish to receive the stream, introducing the possibility of long-term, repeated interactions among them; third, it can provide a sort of “reward” to the cooperating nodes, in the form of better data reception quality and lower playout delay.

Incentives have been mainly studied so far as a way of preventing *freeloading* in distributed systems, enforcing a “fair” retribution. More recent studies [45][75], however, acknowledge the emergence of clustering behavior in incentive-based systems as a secondary phenomenon, which in some cases is deeply linked to the good performance of the system in exam (e.g. BitTorrent [67]). These results support our long-standing intuition about the usefulness of clustering by resource availability as a way to achieve dynamic overlay optimization in P2P unstructured live streaming systems: by allowing nodes rich in upload capacity to position themselves near the source, the higher fan-out they provide can hasten the initial distribution of data chunks. This approach can shorten the average path lengths as it tends to achieve an optimal node placement [103].

3.1.4 Claims

PULSE introduced from its very beginning in 2004 [84] several seminal design choices:

1. **The use of a mesh-based organization of the streaming overlay**, when the only approach that was widely considered viable was based either on trees or multiple-trees.

The first published studies that argued for the viability of mesh-based systems were about DONET/Coolstreaming [122] (2005) and Chainsaw [81] (2005).

2. **The deliberate choice - at all levels - of dynamic system organization over fixed overlay structures**. This choice has very profound implications on the attitude of a system toward churn. Fixed-overlay systems react to churn as an exceptional event, and perform special operations to bring back the system to its 'normal' state. A dynamic system, on the other hand, implies that a certain amount of churn is always present even while it operates 'normally', and its algorithms further contribute to this randomness, paradoxically exploiting it to improve the overall stability of the system. The first published work we are aware of that introduced dynamism from churn as a normal component of the system operation was Chunkyspread [112][113] (2006).

3. **The use of incentives to discourage freeloading**. While incentive-based schemes have received a lot of attention in the last few years, mainly attributable to the blazing success of the BitTorrent [33] bulk data-distribution system, the application of incentives against freeloading to the live streaming context has been slow and not very successful. Several systems have been devised that include some kind of incentive mechanism: for instance, [89] (2004) described a multiple-tree architecture where tit-for-tat was implemented to prevent freeloading. The fundamental issue with such a scheme is that, since exchanges between nodes over several disjoint trees cannot be expected to be reciprocal, it requires an external tamper-proof decentralized debit/credit logging system, that adds another layer of complexity on top of the streaming system itself.

4. **The use of incentives to promote the contribution of resources** by users of the system. More often than not, when incentives have been implemented, their goal was to prevent freeloading by sanctioning nodes that contribute less than expected. Only recently, in [107] (2006), a multiple-tree system is described where nodes are allowed to join a number of trees which is proportional to their resource contribution, while the excess resources are allocated to all the peers without restrictions. Another multiple tree-based system, CROSSFLUX [97] (2006), uses incentives to establish a variable number of backup connections: peers that contribute their fair share of resources are rewarded with a better protection against churn. In the context of unstructured live streaming systems, Chunkyspread [113] (2006) gives the option of using incentives to bias the peer selection process toward those peers that contributed more.

5. **The use of a feedback loop based on the present state of data distribution in the system and on pairwise local incentives to provide dynamical adaptation to both heterogeneous upload capacity and network topology**. The approach we propose in PULSE aims to exploit an underlying network where resources are unevenly distributed: when placed near the source, the nodes contributing excess capacity will greatly reduce the lag perceived throughout the system as if the out-degree of the source was much bigger of what it actually is. On the other hand, peers that contribute less than the stream rate will still be able to participate to the streaming session. This is the essential mechanism that makes the

data-driven topology adaptive to the available network resources. No other systems we are aware of currently use this technique.

In addition to the original design of the application, we also **introduced a series of metrics based on buffer parameters** such as *node lag* (see Chapter 4) in order to describe the instantaneous state of data-driven live streaming systems. The use in the literature of similar metrics first appeared in a 2006 measurement study about PPlive [55].

3.2 Terminology

This section describes the concepts and terminology used throughout the rest of this document. The following pages will refer to definitions and abbreviations contained in Table 3.1.

3.2.1 The Peers

Similarly to all other peer-to-peer applications, a PULSE system is constituted of a multitude of peers (or nodes). We remember from the previous chapter that a *peer* is a software process running on a networked machine: in the following pages, we will use the terms **node** and **peer** interchangeably. As more than one node can be running on the same machine, and as every instance of the program must bind to a different network socket for listening, each peer can be uniquely identified by the tuple <IP address, port>. For brevity, in the following we will use the notation P_i to refer to the unique **network identifier** of peer i. We suppose that each peer i has a certain amount of bandwidth resources. We will call U_i and D_i respectively the upload and download capacity, that is the maximum amount of bandwidth available for sending or receiving data. All the peers that are retrieving the same video stream at the same time are part of the same **streaming session**.

Peer Initialization A PULSE peer needs to obtain some information before it can join a streaming session and receive the media data. This information is usually provided by the publisher of the stream under the form of a short initialization file: we refer to it as the **.pulse** file, from its default file-name extension, e.g. *video1.pulse*. This file contains:

- Metadata about the stream being diffused (title, author, start date, indexing information, etc.)
- A set of one or more identifiers of peers that are part of the streaming session at the moment of the creation of the .pulse file (a.k.a. an **entry point** to the system)
- Information on the parameters of the stream being broadcast
- Information on the protocol parameters that the peers must adopt

- The source's public key for chunk integrity verification
- A signature by the source on the whole content of the .pulse file

The .pulse file is named in the same spirit of its BitTorrent homologous, called *.torrent* file, and it plays a very similar role. The main difference between the content of the two files is that the *.pulse* file does not contain the cryptographic hash of all the blocks of the data that are being distributed. This difference is due to the fact that, in the live streaming context, data are not available a priori to the node that distributes them. For this reason, pollution attacks by resourceful peer that distribute corrupted data are very much a concern for practical live streaming applications [37].

In the general case of live streaming, the checksum information to guarantee the **integrity of the stream** can only be generated *on the fly*, and has to reach the nodes with a low delay, so that data verification (and error recovery) can be performed *before* the chunk is played. In a centralized system, all the peers could retrieve the stream of chunk checksums from an authoritative party, possibly the video streaming source itself. In a completely distributed system, however, such a solution is not acceptable, as it introduces a single failure point in the system. While not a bandwidth-intensive operation, centralized checksum retrieval can limit the scalability of the system.

In a distributed system, the least cumbersome way to guarantee data integrity is using asymmetric cryptography and certificates[5]. The streaming source can specify its public key in the .pulse initialization file, and then digitally sign the contents of the .pulse file (making the .pulse file a sort of self-signed digital certificate[6]). Then, the source can append to each data chunk a cryptographic signature computed with its private key. These chunk signatures can be easily verified by peers using the source's public key.

Membership Management When a new node joins a streaming session, it needs to quickly gather the contact information for a number of nodes already in the system, and should at the same time make other nodes aware of its presence. This problem is referred to in the literature as **membership management**, and can be seen as an independent issue from the core functionality of the PULSE system.

Several examples of membership management can be found in the literature. The simplest way to approach the problem involves the use of a central entity (e.g. the BitTorrent [33] tracker) that keeps track of who is in the system and provides a list of suitable candidates for future interaction to the incoming nodes. Distributed approaches to membership management have been the focus

[5]This section is provided only for completeness, to indicate a possible solution of a related security problem in the distribution of live streams. However, a complete analysis of security concerns lies out of the scope of this work.

[6]If we suppose that the .pulse files are retrieved from a trustworthy source, then a self-signed certificate will be sufficient to guarantee integrity. Otherwise, a trust chain to a well-known certificate authority would be necessary to detect a .pulse file that has been tampered with (e.g. by replacing the source key and signature with someone else's key and signature).

of recent research on efficient primitives for group communication: they involve the creation of a suitable overlay network on which membership messages will travel and reach a sufficient fraction of the system population.

A naïve distributed solution can be based on *flooding*, e.g. similar to the early Gnutella 0.4 protocol: nodes establish a limited number of connections toward their peers - the resulting overlay is thus a "random" graph - and when a message is received by a node it is forwarded over all its connections (except the one on which it was received). To improve the efficiency of this basic solution, other approaches have been suggested that involve either the use of *structured overlays* (e.g. the control tree in Bullet [65], the DHT infrastructure in Splitstream [21]), *clustering* (e.g. the layered cluster infrastructure of NICE [12] and ZIGZAG [110]), or *unstructured gossiping* (e.g. the probabilistic broadcast in SCAMP [48], the random shuffling of neighbors in CYCLON [117], the biased random walks in Swaplinks [115]).

In PULSE, we selected a gossiping approach based on the SCAMP membership management protocol. Randomized gossip offers several advantages, including the good (logarithmic) scaling of the local neighbor list size at each peer - called **partial view** - and the robustness of the resulting connection graph, which can resist high levels of churn. The membership information messages convey summary information on the status of data reception at the individual nodes and are propagated on the SCAMP overlay using fixed-length random walks: we will henceforth call them **BLUE messages**. In the context of membership management, the **loose distributed synchronization** to the source clock is also performed (see 3.2.2 below).

Data/Control Connection Management After a node has gained a coarse grained knowledge about other peers, it starts contacting them directly to obtain detailed information on the contents of their data buffer[7] and to advertise its own buffer contents. The list of nodes about which BLUE messages have been recently received is called the **BLUE knowledge set**. A subset of these nodes is chosen by each peer as the recipient of fine-grained buffer information via PULSE protocol messages, called from now on **RED messages**. The set of nodes selected for this purpose is called **RED neighbor set**. The list of nodes about which fine-grained information has been recently obtained via either solicited or unsolicited RED messages is called **RED knowledge set**.

As the primary goal of any peer is to retrieve useful data chunks and recover a playable media stream, each peer must contact neighbors in the RED knowledge set that are able to provide useful data chunks. At the same time, the node must in turn collect and honor the chunk requests coming from its neighbors. The neighbors that a peer serves are organized in several groups, called **node exchange lists**. These lists are based on the outcomes of the **peer selection algorithm**, which is executed with a fixed periodicity (once per **EPOCH** - which amounts in general to few seconds) and on the behavior of partner nodes. There are three node exchange lists that are served with a decreasing level of priority: the MISSING, NEW, and FORWARD lists. Their complete definition is deferred to Section 3.3.

[7]The buffer collects and stores the received data chunks before they are sent to the player application.

Parameter	Typical Value	Description
W	32	Length of buffer sliding window [*chunks*]
TW	64	Total length of trading window [*chunks*]
SBR	256	FEC-encoded stream bit rate [$Kbit/s$]
R	16	Rate of chunk generation @source [s^{-1}]
LR_{max}	25%	FEC tolerance to chunk losses/window
S	$LR_{max} \cdot W$	Window sliding tolerance [*chunks*]
T_D	$\frac{250}{R}$	Min. node lag to trigger buffer reset [*s*]
EPOCH	2	Time b/w subsequent peer selections [*s*]

Table 3.1: Summary of System-Wide Parameters

3.2.2 The Stream

The stream is a sequence of data that is generated by an audio/video source (e.g. camera, microphone, recorded media file). The stream is encoded using one or more[8] **codecs**, video/audio compression algorithms that can take several parameters to produce a coded output with the desired properties. Codecs are defined **lossy** if information is irreversibly lost during the encoding phase, **lossless** otherwise. For the practical purposes of video transmission over narrowband channels, lossy codecs are the preferred choice as their bandwidth requirement is lower.

The codec parameter that is most relevant to our discussion is the **nominal bit-rate**, which determines both the video/audio quality and the bandwidth requirement of the video stream. Lossy codecs are further divided in two categories: constant bit-rate (**CBR**) and variable bit-rate (**VBR**), depending on whether the bit-rate of a stream remains unchanged over time or adapts to the features of the video data: while the nominal bit-rate of a VBR codec can continuously change, the average bit-rate achieved when compressing a given media sequence is constant.

Streaming Source and Data Chunks In PULSE, all peers are identical pieces of software implementing the same algorithms. However, one peer in each streaming session needs to behave differently: this special peer is the **streaming source**, and is the only peer that introduces new data from the stream into the system.

The streaming source receives an encoded video stream as its input and turns it into a sequence of **data chunks**. The chunks are the basic unit of data exchange in the system and are generated by the source with a fixed rate R, the **chunk rate**. Chunks contain, beside the video data, several pieces of control information, such as the chunk **sequence number** c_j, a counter which is incremented by one for each chunk, and the **chunk timestamp** $\tau(c_j)$, the time at which the chunk j was generated. This timestamp is generated using the local clock at the source, also called the **media clock**.

[8]Each component of the stream (audio, video) is encoded separately using specialized codecs. Then, the audio and video substreams are packed together, with the possible addition of metadata, padding, and redundant information for error recovery using a *container format*.

While the chunk sequence number is mostly used for internal node operations (ordering the chunks in the buffer, finding the missing chunks), the chunk timestamp is used by a peer P_i to evaluate the "age" of a chunk c_j upon its reception: we will call this delay **chunk lag**, defined as $T_{c_j}(t) = t - \tau(c_j)\ \forall t > \tau(c_j)$. The use of lags (i.e time delays) that are relative to the generation of chunks at the source will be useful for the purpose of representing the status of the system at steady state, as we will better explain in the following pages.

Chunk Loss Recovery The streaming source can also process the video data and add information to enable the recovery of a certain amount of chunks that (for some reason) have not reached a node. In PULSE, we adopt a chunk-level forward error correction (FEC) technique based on the classic *(N,K) Reed-Solomon coding*, which has often been used in the literature for similar purposes [58][93][78].

The source applies FEC coding[9] to protect the stream from up to a specified **loss rate** (LR). Coding is performed by first splitting the original video stream into a block of K chunks, and then generating $S = W - K$ linearly independent parity chunks[10], for a total number of W chunks in each block: the coding rate is thus equal to $\frac{K}{W}$. After this process, the W chunks are made available to the nodes in the system: each node will only need to retrieve any $K' \geq K$ chunks to be able to recover the original video stream data. To guarantee the full recovery of the stream, W and K will be chosen so that the rate of *redundant coding* is sufficient to compensate the maximum amount of expected losses, i.e. $1 - \frac{K}{W} = \frac{S}{W} \geq LR_{max}$. LR_{max} is called the **maximum tolerable loss rate**, a system parameter that depends on the expected operational conditions of the streaming application.

3.2.3 Receiving the Data

The streaming source delivers the most recent data chunks to the nodes it is connected to in a **push-based** fashion (only considering their buffer information to avoid sending duplicates). On the other hand, ordinary peers request (**pull**) the missing chunks from their neighbors. Using the collected buffer information from red messages, nodes can issue **chunk requests** for one or more individual chunks to the neighbors in the RED knowledge set. Chunk requests are assigned to the nodes according to the **request scheduling algorithm**.

After receiving a chunk request, a peer stores it in the appropriate per-neighbor **request queue**. Chunks are served to nodes in a priority-based round-robin order over the various exchange lists (MISSING, NEW and FORWARD): for each node, the chunks are chosen using the **sender scheduling algorithm** among those that were requested, until there are no more chunks to be sent or the bandwidth resources are fully utilized.

[9]For the sake of maintaining coherence with the notation in the next pages, the (N,K) parameters of the Reed-Solomon coding are henceforth renamed (W,K). Incidentally, W is used because the total number of chunks in a FEC block will be equal to the length of the buffer sliding window of the PULSE node.

[10]Incidentally again, S will be defined as the *sliding tolerance* of the node buffer window, i.e. the number of chunks that a peer can skip during data reception without compromising its playout quality.

Buffer Size and Data Reception An exchange of data between node *A* and node *B* can be performed when *A* has in its buffer one or more chunks that *B* has not obtained yet: we define this condition as **buffer overlap**. Since every peer keeps the chunks it has received in its buffer for only a short time and is free to discard them once they have been sent to the player, the overlap between the buffers of the two nodes is null or finite, and its maximum extent is determined by the size of the node buffers. The synchronization of the node buffers has to be encouraged to obtain higher data transfer rates between nodes, as the overlap is maximum when the two buffers are perfectly aligned. Synchronization increases at the same time the likelihood of bi-directional exchange.

The two basic conditions of live streaming, i.e. the continuous generation of new chunks by the source and the need of their sequential delivery to the application, do not imply an upper bound on the node buffer size: in fact, the larger the buffers, the higher the chance that data transfers will be possible between two nodes. On the other hand, the additional constraint of near-optimal playout latency introduces an upper bound on the size of the buffer: the larger the buffers, the longer the time required for a complete series of chunks to be retrieved and passed to the player application. Therefore, the choice of a system-wide buffer size also involves a trade-off between playout latency and throughput.

3.3 Structure of the PULSE Node

The PULSE node contains a set of data structures and methods that manage the different aspects of its behavior: the **Data Buffer**, which collects, reorders and transfers data chunks to the player application, the **Knowledge Manager**, which organizes the information that a peer holds about the rest of the system, and the **Trading Logic**, which executes the algorithms that determine the node behavior using the information coming from the Data Buffer and the Knowledge Manager.

3.3.1 Data Buffer

The buffer of a PULSE node is more than a simple data structure that stores the data chunks prior to their delivery to the application. Actually, it is a sophisticated object that performs many operations, and that actively influences the behavior of the node:

- It determines on *which set of chunks* the chunk selection algorithm will operate, based on their playout priority and on the current buffer content.
- It influences key aspects of the peer selection algorithm, as the profitability of a relationship with other remote nodes depends on the presence of overlapping data ranges.
- It guarantees the integrity of the data stream sent to the player, exploiting the FEC encoding performed by the source on the data chunks.

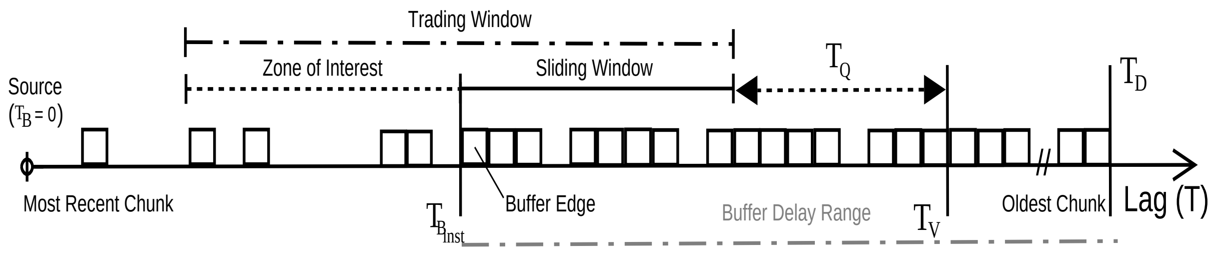


Figure 3.1: The Buffer of a PULSE Node

- It measures the data reception quality, and enables the node to timely react to transient reception shortages.

Structure The PULSE buffer is designed to decouple the data retrieval process from the delivery of the stream to the player (Figure 3.1).

The data exchange involves only a limited portion of the buffer, called **Trading Window**, which contains $2W$ contiguous chunks. The first (lower) half of the Trading Window, which contains chunks with smaller sequence numbers (i.e. older), is called **Sliding Window** and contains W chunks. The other (upper) half, which contains chunks withs bigger sequence numbers (i.e. more recent), is called **Zone of Interest**. The most recent chunk in the Sliding Window is called **buffer edge** and is referred to as c_β.

The chunk lag of the buffer edge is used to represent the current reception status of the node: the **instantaneous node lag** is defined as $T_{B_{inst}}(t) = T_{c_\beta}(t) = t - \tau(c_\beta)$. For instance, the availability of chunks in the Trading Window is periodically advertised by every node to its neighbors in the form of a bitmap of $2W$ bits, along with the sequence number of the buffer edge c_β.

The delivery of the data to the application is performed after the data chunks reach a lag equal to T_V, called **playout delay**. The value of T_V is decided during the initialization phase and, once set, it is bound to remain constant during the whole streaming session - unless the node experiences severe reception shortage, as we will explain later. Once passed to the application, data can be discarded by the node, to limit the memory usage by the buffer: the lag value at which chunks are discarded is called T_D. This lag value also conventionally marks the **reconnection threshold** of the node buffer, as the chunks are assumed to be too old for playout.

Operation A parameter $S = W - K$, called **sliding tolerance**, defines the minimal amount of chunks that can be missed by the sliding window while it moves forward. The maximum chunk loss rate tolerated during normal peer operation is thus bound by $LR = \frac{S}{W}$. The system-wide parameter LR_{max} is equal to the amount of redundant coding performed by the source. The value of S at every peer must be set so that $LR \leq LR_{max}$ to ensure the complete recovery of the original data stream. If less than K chunks are available, the sliding window cannot move. The lag of all the chunks contained in the window increases as time passes and as new chunks are generated. Otherwise, the window will keep sliding forward as long as it contains at least K chunks.

Let's suppose that at time t the buffer is operating at steady state, well after initialization has been performed: the buffer contains a long continuous sequence of chunks, the node is requesting chunks in its Trading Window zone, and the Sliding Window contains K or more chunks. The **buffer delay range** is defined as the sequence of chunks that spans from T_V to $T_{B_{inst}}(t)$.

Because of the randomness of the data retrieval process, the value of the instantaneous node lag $T_{B_{inst}}$ fluctuates over time in an unpredictable way. At time t_0, when data chunks are received and chunk c_i becomes the new buffer edge, the node lag value is $T_{B_{inst}}(t_0) = T_{c_i}(t_0) = t_0 - \tau(c_i)$. While chunk c_i is the buffer edge, the value of $T_{B_{inst}}$ linearly increases with the time, i.e. $T_{B_{inst}}(t) = T_{c_i}(t_0) + (t - t_0)$. When chunk $c_{i+k} \quad \forall k > 0$ becomes the new buffer edge at time $t = t_1 > t_0$, the value of node lag suddenly decreases: $T_{B_{inst}}(t_1) = T_{c_{i+k}}(t_1) = t_1 - \tau(c_{i+k}) < T_{c_i}(t_0) + (t_1 - t_0)$.

As new data chunks are introduced into the system at a constant rate, it is useful to study the evolution of instantaneous node lag over time. In the example above, while the value of $T_{B_{inst}}$ keeps changing, it is possible to make a short-term estimate of whether its average value is increasing or decreasing (that is, whether its derivative is positive or negative). Since the chunk generation rate is R, we know that chunk c_{i+k} was first distributed by the source $\frac{k}{R}$ seconds after the chunk c_i. We can then compute the $T_{B_{inst}}$ differential between times $t_0$and t_1 as:

$$\Delta T_B(t_0,\, t_1) = \frac{T_{c_{j+k}}(t_1) - T_{c_i}(t_0)}{t_1 - t_0} = \frac{(T_{c_i}(t_1) - \frac{k}{R}) - T_{c_i}(t_0)}{t_1 - t_0} =$$

$$= \frac{(T_{c_i}(t_0) + (t_1 - t_0) - \frac{k}{R}) - T_{c_i}(t_0)}{t_1 - t_0} = 1 - \frac{k}{R\,(t_1 - t_0)}$$

which is positive when $\frac{k}{t_1 - t_0} < R$, i.e. lag increases because fewer chunks have been received than the source has generated during the $[t_0, t_1]$ time interval, and negative when $\frac{k}{t_1 - t_0} > R$. If no new chunks are received, $T_{B_{inst}}$ grows over time at a constant rate, and the window keeps drifting on the lag axis (to the right in Figure 3.1) with a constant speed. Only when at least K chunks over W have been collected, the window is allowed to slide and to reduce its $T_{B_{inst}}$ (moving to the left in Figure 3.1).

A Safety Margin between Reception and Playout Chunk $c_\beta - W$ is the most recent chunk that can be played by the local node without any loss of quality[11]. We call T_Q the interval that

[11]As FEC encoding is performed on blocks of K chunks and produces $W = K + S$ chunks, the fact that all chunks up to $c_\beta - W$ have passed through the sliding window (with less than S chunks lost per window) guarantees

spans from the end of the sliding window to the playout delay: $T_Q(t) = T_V - (T_{B_{inst}}(t) + \frac{W}{R})$. Continuous playout without quality degradation requires T_V to remain constant during the whole streaming session: the duration of T_Q indicates the **margin of safety** that the system maintains against data reception problems, as at least T_Q seconds of playable media data have already been retrieved by the peer. During system operation, it may then happen that T_Q drops to zero or becomes negative due to an extensive reception shortage (i.e. when $T_{B_{inst}}(t) + \frac{W}{R} \geq T_V$). At this point, no more data can be copied from the buffer to the player without disrupting the integrity of the stream.

If shortage persists, $T_{B_{inst}}$ may grow up to T_D, at which point the chunks the node is requesting are no longer available in the system: the node is then forced to reset its $T_{B_{inst}}$ value and repeat the initialization procedure (detailed below). Abrupt changes in the duration of T_Q (while it is still positive) can also be used by peers to predict impending reception problem: nodes can leverage this information to preemptively react to avoid playback disruption[12]. The consequence of reception shortage that are experienced by the user is a temporary interruption of playback, with the loss of a segment of media stream.

The (Average) Node Lag We define the **node lag** T_B as the average of the instantaneous node lag. The value of $T_{B_{inst}}$ is sampled by each node at regular time intervals of γ seconds. The average node lag can then be computed either as the simple moving average (SMA) on a finite window of the last n samples of $T_{B_{inst}}$ or as the exponentially-weighted moving average (EWMA) over the whole sequence of samples. If the EWMA notation is used, node lag is defined as:

$$T_B(t) = (1-a)\,T_{B_{inst}}(\,(kn-1)\gamma\,) + a\,T_B(\,n\gamma\,) \qquad \forall t \in [k\gamma,\,(n+1)\gamma)$$

If we use the SMA notation, node lag computed over the window of the n most recent instantaneous lag samples can be written as:

$$T_B(t) = \frac{1}{k}\sum_{i=0}^{n} T_{B_{inst}}(i\gamma) \tag{3.1}$$

Node lag is used to advertise the approximate range of chunks that a node is able to provide. Messages containing the values of T_B and T_D provide a synthetic and durable description of the data content of a node: the choice of new partners for data exchange can be based upon this information, as it is the result of a long-term averaging process.

that lossless playout can be achieved *at least* up to chunk $c_\beta - W$.

[12]The current implementation of PULSE does not include any reactive policy to anticipate or avoid buffer reinitialization.

3.3.2 Knowledge Management

The Knowledge Manager organizes the information a peer has about the rest of the network. Its role is very important in an unstructured system, as most node decisions are based on the currently available local knowledge. The information stored at peer P_i includes:

- direct measurements of network parameters (RTT, data throughput per connection)
- information about the *network address* and buffer delay range of the other peers.
- detailed accounts of the exact buffer content of the remote peers, required to engage in data exchange.
- local records of previous trading interactions, in the form of a cumulative *history score* H for each of the peers that have had previous interactions with peer P_i.

Messages Nodes exchange two types of messages, BLUE messages and RED messages. **BLUE messages** carry summary information about the data available at the node in the form of the extreme lag values of the $[T_B, T_D]$ interval in its buffer: these values do not refer to specific chunks, but give a durable notion of which data range a node can be expected to provide, as the values of $T_B$and T_D are expected to remain stable when the system operates at steady state. These messages are forwarded among all peers using a protocol, such as random walks with a finite length [49]. **RED messages** carry a timestamp, the instantaneous node position $T_{B_{inst}}$, T_D, the bitmap summarizing the chunks present in the Trading Window, and (optionally) explicit request bitmaps for data chunks in that range. These short messages are directly exchanged between pairs of nodes, and it is possible to use them to estimate the pairwise network latency. Two additional messages, **HELLO / HELLO REPLY messages**, are only used when a node joins the PULSE system to perform its insertion in the membership management overlay.

The Node Lists While the *network addresses* of all the nodes that have been contacted during the life of a peer are stored in a list, similar to the *host catcher* of a Gnutella node, only a small number of nodes is contacted on a regular basis during the operation of PULSE in order to obtain up-to-date information. Two lists are actively maintained to support the needs of the streaming application:

The BLUE Knowledge List - PULSE relies on an unstructured substrate to guarantee that the system will stay connected and that every node knows and can contact enough peers. The substrate currently implemented by PULSE is based on SCAMP [47], a randomized gossip membership management service that shows good asymptotic properties, such as the logarithmic growth with the size of the system of the *partial views* at each node. *BLUE* messages are periodically exchanged over this substrate: upon receipt of these messages, the local peer records their content in the BLUE knowledge list. This list currently has a fixed size[13] (several tens of nodes) and is regularly purged from older records as new messages come in.

The RED Knowledge List - To be able to perform data exchanges, a PULSE node requires an up-to-date view of the buffer content of a subset of the system population. A number of peers that sent a recent RED message are included in this list, whose maximum size is also currently fixed to few tens of peers. To populate its own RED list, every node periodically chooses a small number of peers from its BLUE list and sends them a RED message about its current situation, without any chunk request. Also, a node responds with a RED message (that may or may not contain chunk requests) to the first RED message without requests it receives from nodes not present in its RED knowledge list.

3.3.3 Trading Logic

Peer selection algorithms determine how the associations between nodes are established and maintained. In the global context of distributed systems, the problem of selecting a subset of the peers to exchange data and control information appears constantly: the role of peer selection can be more or less relevant depending on the applicative requirements, the operating environment, and the architectural choices made by the designers of a system. Live streaming systems, for instance, require the rapid propagation of data across the node population. The three prevalent architectures for live streaming - overlay-driven, multiple-tree based, and data-driven - all use selection algorithms with different characteristics and purposes.

In *single-tree systems*, the purpose of peer selection is to incrementally build (and, when needed, repair) a good overlay tree. The choices on the placement of incoming nodes are thus often performed by the nodes already in the system, and the join procedure may involve the streaming source or an external rendez-vous point [12]. The reason is that internal nodes have more information on the current overlay topology than the new peer, and can optimize the placement of nodes for several criteria such as lowest latency and highest bandwidth availability. In *multiple-tree systems*, the main purpose of node placement is to guarantee that all nodes have an uniform upload distribution. Additional goals can be overlay path diversity [21], improved resilience [80], incentive support [97] and resource-adaptive load balancing [113]. The placement can be performed using an additional structured overlay (such as a DHT), relying an external rendezvous point or with the help of nodes already in the system. In *mesh-based systems*, the peer selection algorithms are even more important, as, unlike trees, meshes do not provide guarantees on the flow of data across the system. This means that each node has to choose - autonomously and periodically, based on a steady exchange of control information - the peers it should associate with. A chunk scheduling policy is also required in the mesh-based case to insure the timely and steady delivery of the stream data.

The trading logic of a PULSE node controls all the aspects of chunk request, selection, and scheduling. It processes the information coming from both the local buffer and the knowledge

[13]The maximum size of the BLUE knowledge list could be adapted to the node population size based on the estimates performed by the SCAMP substrate.

Name	Value	Meaning
δ	$\frac{3}{8}W$	Offset for initial chunk requests [*chunks*]
$T_{Q_{INIT}}$	-	Margin of safety at buffer initialization [*s*]
MAX_MISSING_SLOTS	4	Peers chosen w/ optimistic TFT incentive
TFT_RESERVED_SLOTS	3	Max slots for pure TFT
MAX_FORWARD_SLOTS	8	Peers chosen for altruistic relationships
R_{TOT}	16	Max total outstanding requests
R_{max}	2	Max outstanding requests to same peer
H	-	A remote node's *History* score
C	-	Latency bias for opt. peer selection
latency		Measured latency to remote node
NEARBY_THRESHOLD	$\frac{2W}{R}$	Minimum ΔT_B to select P_i as FORWARD
FAR_THRESHOLD	$\frac{4W}{R}$	Maximum ΔT_B to select P_i as FORWARD

Table 3.2: Other Parameters Appearing in the PULSE Algorithms

logic to decide which and how many chunks will be requested/sent from/to which neighbor. Its two main components are the *Peer Selection* and the *Chunk Scheduling* algorithms:

- The Peer Selection module maintains three exchange lists, which contain the peers that can receive service by the local peer during the current EPOCH. These lists, MISSING, NEW and FORWARD, contain nodes chosen with different criteria and are receive service with different priorities.

- The Chunk Selection module decides which chunks will be requested from which node from the exchange lists, based on the information contained in the RED knowledge list.

The rationale and the algorithmic details of the trading logic will be explained in the next section.

3.4 Algorithms

This section describes the algorithms used by a PULSE peer, from its first connection to the network to steady state operation. We will present the join phase, when incoming nodes enter the system and synchronize with the source clock, the initialization phase, when the node receives the first data chunks and chooses its buffer parameters, the data exchange phase, where peer selection and chunk selection are performed regularly, and the recovery phase, in which the nodes overcome the effects of a permanent download starvation. Table 3.2 summarizes the parameters that will appear in the following pages.

Algorithm 1 Buffer Initialization: Condition to Set Initial Window

```
for idx in range $(\delta + \frac{W}{2},\ \delta - \frac{W}{2},\ -1)$:    # idx is the lag of a virtual sliding window vsw(idx)
    if ||chunks in the SW of vsw(idx)|| $\geq \frac{W}{4}$ AND ||chunks in the TW of vsw(idx)|| $\geq W$:
        set $T_{B_{INIT}} = T_{c_{idx}}$
```

3.4.1 Joining the Network

To join a PULSE session, the incoming peer has to know the IP address and port of at least one node that already belongs to the system. This information can be supplied by any *bootstrap mechanism*: in the current implementation, a file that contains the addresses of one or more nodes (not necessarily including the source node) is used as the starting point for node bootstrap.

The joining peer P sends HELLO messages to the bootstrap nodes. The HELLO messages are propagated throughout the network using the SCAMP algorithms [47] until they reach a peer that can accept the new neighbor: this peer will then directly send P an HELLO REPLY message and will add it to its local view. SCAMP guarantees that, for large node populations, the number of nodes that will have P in their local partial view - and also the size of the partial view of P - will be in the order of $O(\log N)$.

3.4.2 Initializing the Buffer

After a node has joined the system, it begins requesting chunks in the $[0+\delta,\ W+\delta]$ fixed delay interval[14]. As chunks are retrieved, they are put into the buffer: in this initial phase, the sliding window is not yet enabled, and chunks are requested with the goal of forming a nearly-contiguous block. At t_0, when at least $\frac{W}{2}$ over W chunks have been collected in the Sliding Window and W over the whole Trading Window (Algorithm 1), the sliding window mechanism is first enabled around that block of chunks, and the initial buffering lag $T_{B_{INIT}} \equiv T_{B_{inst}}(t_0)$ value is set to the lag of the current buffer edge. As soon as the window contains enough chunks to begin sliding forward - unless the value of T_B meanwhile grows too large, in which case the initial window is cleared - the buffer keeps operating as described above (Figure 3.1). The play-out delay T_V is determined after time t_0, when the node buffer at least contains a continuous sequence of $R \cdot T_{Q_{INIT}}$ chunks. Only at time $t_1 > t_0$, when $T_{B_{INIT}} - T_{B_{inst}}(t_1) = T_{Q_{INIT}}$, the media play-out is allowed to begin. T_V is then set at time t_1 to be equal to $T_{B_{inst}}(t_1) + T_{Q_{INIT}} + \frac{W}{R}$.

3.4.3 Bandwidth Allocation

At any given moment, each peer maintains several connections for sending and receiving data. To simplify the problem of bandwidth allocation, PULSE peers try to establish a fixed number of

[14]The δ parameter introduces an offset in the initial lag interval where incoming nodes request their first chunks. Requesting chunks with higher δ may increase the speed of node initialization (when resources are available) as older chunks are usually better replicated in the system.

outbound connections for data exchange, but do not explicitly limit the number of incoming ones. As node bandwidth is typically asymmetric, with the upload bandwidth being much smaller than the download bandwidth, it is mainly important to control the number of outbound connections.

The biggest challenge for the bandwidth allocation mechanism is the need to support upload bandwidth heterogeneity: especially in a live streaming application, it is critical to make all nodes contribute, since unused upload bandwidth reduces overall system capacity. Opening multiple connections has two benefits: a node is able to provide service to several peers, and it obtains more information to support its future exchanges. However, the more connections, the higher is the control message overhead for each node. Also, when the upload bandwidth can vary widely, it is difficult to set a fixed ’number of connections’ parameter that works for all the nodes in the system.

In PULSE, we approach the bandwidth allocation problem in a practical way. Nodes populate their MISSING and FORWARD exchange lists on every EPOCH using the peer selection algorithms. A third list contains all nodes that have sent one or more chunks during the current EPOCH and do not belong to the two previous lists: we refer to this list as the NEW list. The only difference in bandwidth allocation between the three exchange lists is that *chunk requests received from peers in the* MISSING *list are honored with higher priority, followed in order by peers in the* NEW *list and peers in the* FORWARD *list.*

On the Number of MISSING Connections We believe that a large number of MISSING connections can reduce the effectiveness of the tit-for-tat selection: we must in fact remember that at steady state, for a *rate-limited* application such as live streaming, no more than SBR bytes per second will be received on average by each node. To clarify this point, let us suppose that MISSING connections alone are sufficient to sustain the reception by a peer of the full stream, and that each peer opens exactly n MISSING connections to other nodes: each node will be selected on average by n peers as MISSING partner. Intuitively, a large n means a lower expected throughput from (and to) each MISSING connection: as the contribution threshold required to gain a place in the MISSING list of the remote node becomes lower, associations become more random and less related to the actual resource availability at the nodes and system performance may suffer because of repeated sub-optimal choices. For this reason, we decided to use a small number of connections (e.g. four) to MISSING partners, so that each peer can expect a meaningful theoretical throughput on each connection (e.g. SBR/4).

On the Number of FORWARD Connections On the other hand, especially for the richer nodes, opening more connections could improve the odds of finding useful chunks and fully exploiting their upload capacity. To take this fact into account, a variable number of connections can then be assigned to FORWARD exchanges, depending on the available upload. We remember that maintaining an open connection to some node does not imply that it will be used for data exchange, as that is determined by the chunk scheduling mechanism: however, these connections may allow resourceful peers to utilize their excess bandwidth to the system by providing a large number of other peers with recent chunks.

Algorithm 2 MISSING Selection Algorithm

```
Old_Contributors   # list of nodes that contributed during last EPOCH
TFT=[]             # an empty list
MISSING_list=[]    # list that will contain the MISSING peers

Sort Old_Contributors by decreasing data_contribution
for each node in Old_contributors:
    if node's data_contribution > MIN_TFT_CONTRIB:
        add node to TFT

order TFT by decreasing data_contribution

while TFT is NOT empty AND length of MISSING_List < TFT_RESERVED_SLOTS:
    take first element out of TFT, called peer
    add peer to MISSING_List
```

3.4.4 Peer Selection

Peer selection is periodically performed by each node, its time period is called *EPOCH* and is constant. PULSE uses two peer selection algorithms: an optimistic tit-for-tat selection based on the total amount of data received during the previous EPOCH (similar to BitTorrent) and a lag-constrained selection based on a cumulative trust score. The two algorithms are executed at the start of each EPOCH, and give as a result two lists of peers, the MISSING and the FORWARD list.

MISSING Selection: Optimistic Tit-for-Tat

This policy aims to identify which peers, among all those about which a node has knowledge, are currently interested and able to provide data in the short term. Two pieces of information are 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 MISSING list (Algorithm 2). 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' and improve the support for topological locality: the latency-aware optimistic selection algorithm is shown as Algorithm 3.

Algorithm 3 Optimistic Peer Selection Strategy

```
Candidates, Overlap=[]        # empty lists

Candidates ←{ n | n∈RED_Knowledge_List AND∉MISSING_list }
Overlap ← { n | n ∈ Candidates AND have overlapping TW }

# latency(x) returns the network latency between of node x measured by the local peer
# $C$ is the latency bias
for each peer in Overlap:
    Compute Distance=abs($T_B$(local)-$T_B$(peer))+$C$·latency(peer)
order Overlap by increasing Distance

while Overlap not empty OR length of MISSING_List < MAX_MISSING_SLOTS:
    remove first node from Overlap
    add node to MISSING_List
```

FORWARD Selection: Round-Robin on History Score

Every node maintains a record of the previous interactions with every other peer as a numeric value, which we called the **history score** H. This mechanism enables a peer to use data on past behavior of its fellow peers to make *informed choices* when selecting future candidates for FORWARD exchanges. The history score is computed as follows: each time a previously unknown peer is encountered, it is given an initial score. The score is incremented by a fixed value whenever useful chunks are received from a node while it does not belong to the MISSING/FORWARD lists. The 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.

As it is currently defined, the history mechanism can appear rather simplistic, but it proved effective to evenly distribute altruistic contributions among the peers. We believe that the original incentive model proposed by GnuNet [51] could eventually be applied to our system, further improving the strength of the relationships among resourceful nodes.

3.4.5 Chunk Selection and Request

A good chunk selection strategy is one that distributes the chunks in an uniform way across the nodes to avoid situations where some chunks are much less replicated system-wide than others. It should also ensure that the buffer content of nearby nodes is different enough that they can engage in mutual transactions and concurrently exploit their multiple connections. Finally, it should prevent that several neighbors concurrently send duplicate chunks to the same node.

Chunk Selection: Sending The chunks to be sent over a connection, regardless if MISSING or FORWARD, are selected comparing the requests received from each peer to the chunks currently

Algorithm 4 FORWARD Selection Algorithm

```
Candidates, History=[]        # empty lists
FORWARD_list=[]    # list that will contain the FORWARD peers

Candidates ←{ n | n ∈ RED_Knowledge_List AND ∉ MISSING_list }
History ←{ n | n ∈ Candidates AND $H \neq NULL$}

Sort History by decreasing $H$

while History not empty OR length of FORWARD_list < MAX_FORWARD_SLOTS:
    remove first node from History
    Compute Distance=abs($T_B$(local)-$T_B$(node))
    if NEARBY_THRESHOLD < Distance < FAR_THRESHOLD:
        add node to FORWARD_List
```

Algorithm 5 Chunk Scheduling at the Sender Peer

```
ordered_list ← list of chunks in the buffer sorted by decreasing replica_count

for each peer in MISSING_list, NEW_list, FORWARD_list:
    if peer has outstanding requests:
        least_sent ←chunks from requests with the smallest replica_count

        if length(least_sent)>1:
            randomly shuffle least_sent
        chunk=least_sent[0]
        send chunk to peer
        update ordered_list
```

held in the local buffer (Algorithm 5). Requested chunks that are available are then sorted using local ordering criteria, and the first one is chosen for sending. The criterion we are currently using for ordering chunks at the sender is a "Least Sent First, Random" strategy. Each peer keeps a counter of how many times it has sent each requested chunk (*replica count*). The one that has been sent the least number of times is chosen to be sent first. In case of a tie, the chunk is selected randomly. It is indeed possible to queue several chunk uploads toward the same peer to benefit from the effects of transfer pipelining.

This scheduling strategy shows fairly good results, since the newest (and thus rarest, from the point of view of the sender) chunks to be received are among the first that are sent. Breaking ties with a random choice, instead of e.g. selecting the chunk whose lag is lowest, aims to avoid the preferential replication of a same single chunk which may happen in situations where several peers have their trading windows synchronized.

Algorithm 6 Chunk Scheduling at the Source

```
for each peer in SOURCE_list:
    ordered_list= sort chunks by number of times they have been sent
    least_sent= select from ordered_list the least-replicated chunk(s)

    if length(least_sent)>1:
        random shuffle least_sent
    chunk=least_sent[0]
    send chunk to peer
```

Chunk Selection: the Source The streaming source adopts a sender scheduling mechanism which differs from the normal nodes by the fact that the source discards all the requests it receives from other nodes (Algorithm 6). Thus, the source is the only peer in the system that adopts a push-only scheduling strategy, i.e. a random round-robin over the newest chunks.

Chunk Selection: Requesting The algorithm for chunk requests is similar to the heuristic used in DONet/CoolStreaming [122]. Its purpose is to request the rarest chunks among those that are locally available, and to distribute the requests across different possible providers (Algorithm 7). Using the local knowledge gathered from the current neighbor set, chunks that are rarest across the neighborhood are requested with higher priority than more common ones. To limit the load on any single peer, the maximum number of per-node requests is bounded.

3.5 Implementation

The first working version of the PULSE prototype node has been developed during the summer months of 2006 by Diego Perino, who wrote a detailed account of his internship activity in [82]. The node was implemented following the PULSE simulator code as a reference for the data exchange algorithms, while most data structures had to be redefined and the interface to the network had to be designed from scratch.

The internal organization of the node is depicted in Figure 3.2. The subdivision into classes closely follows the scheme, and is also remarkably similar to the system diagram presented in [122]. The solid arrows in the picture represent the flow of control information and data inside the node, while the dotted arrows describe the interconnections between system modules. When compared to the protocol description above, the trading logic and knowledge manager functions are jointly implemented by a *peer manager*, performing peer selection, and a *chunk scheduler*, performing chunk selection, that rely on a common set of data structures. The central block, *system management*, coordinates the timely execution of the various algorithms.

Algorithm 7 Chunk Request Scheduling at the Receiver Peer

```
chunks_needed=[]            # list of chunks the local peer needs (from its TW)

for each peer in RED_Knowledge_List:
  for chunk in chunks_needed:
    if peer can offer chunk:
      add peer to chunk.providers

Order chunks_needed by chunk availability, with n_available as the index:
ordered_same_rarity (n_available, [chunk ID, [providers]])
Sort ordered_same_rarity's lines from the less available set of chunks (rarest first)

for each row of ordered_same_rarity:
  chunk_vec←vector of tuples [chunk ID, [providers]] with highest rarity
  while chunk_vec is NOT empty:
    selected_chunk←remove a random chunk from chunk_vec
    providers=providers for selected_chunk

    while providers is NOT empty:
      sel_provider←remove a random element from providers
      if (sel_provider.requests$\leq R_{max}$)
        add selected_chunk to sel_provider.requests
        sel_provider.requests+=1
        if num_requests$= R_{TOT}$
          return
```

3.5.1 Practical Details

The PULSE prototype is written in Python, an object-oriented scripting and programming language that is well suited for quickly implementing simple applications. One of the main advantages of using Python for our purposes is the availability of the Twisted[15] framework, a library that provides an event-driven networking back-end that takes care of several low-level issues (multiplexed socket accesses, management of network buffers, etc.). The prototype node also uses some external libraries and modules, notably a Python wrapper to the fast C++ Reed-Solomon FEC implementation by Luigi Rizzo [93]. Furthermore, the node implements SCAMP as it is defined in [47] (with the exception of the indirection mechanism). Some other minor modifications (such as a message ID field) were required to suppress repeated loops in scenarios with small node populations, as the use of variable-length random walks with probabilistic termination criteria was generating a large amount of network traffic each time a node would join a PULSE session.

[15]Twisted by Twisted Matrix Labs, http://www.twistedmatrix.com/

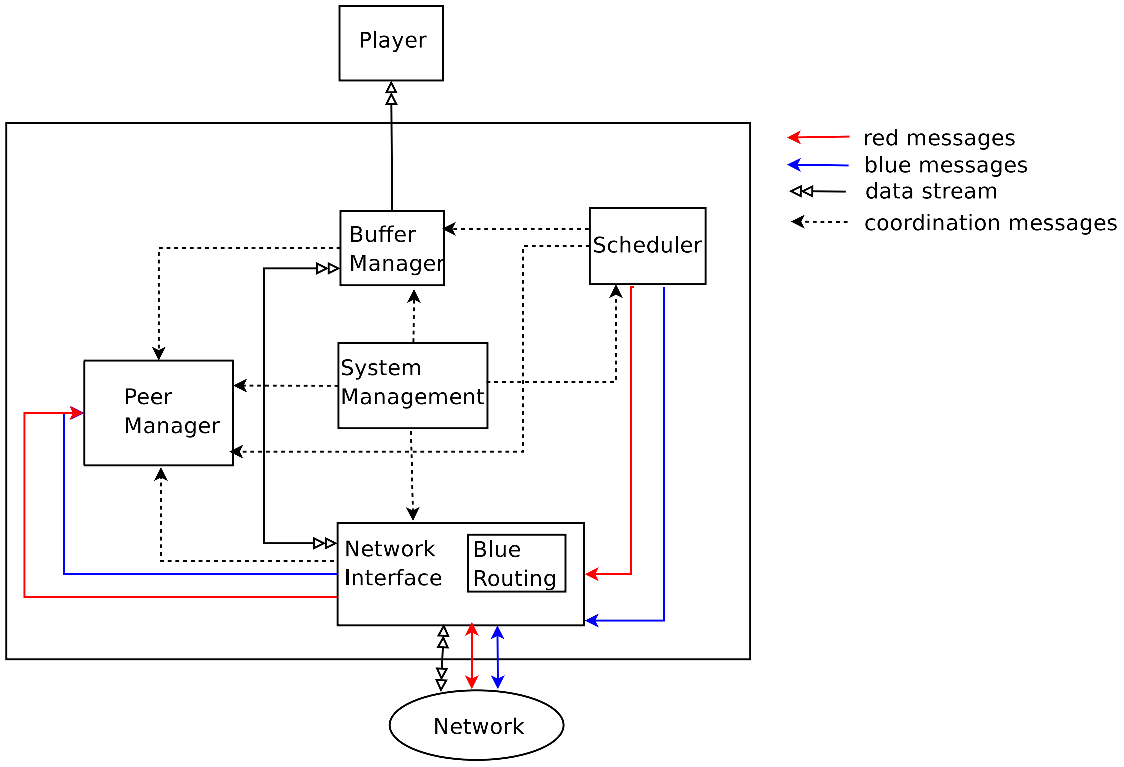


Figure 3.2: Internal Organization of the PULSE Node and Data Paths

Choice of Transport Protocols The PULSE node maintains two transport sockets: a TCP socket for data transfers and an UDP socket for RED and BLUE control message exchange. The choice of TCP connections for data transfers is motivated by the need of reliability, as data chunks span over several IP datagrams, making an unreliable transport protocol such as UDP unsuitable for the task. PULSE can adopt rather large chunk sizes and small chunk rates in virtue of the less stringent timeliness constraints of live streaming compared to interactive video distribution. For instance, typical chunk sizes will range from few to several tens of kilobytes, while chunk rates will range between 2 and 16 chunks per second. We remember that, given the chunk rate, chunk size can change between PULSE sessions as it depends on the bitrate of the stream being broadcast.

The use of UDP is motivated by the fact that loss of control messages is tolerable and does not need recovery. The exchange of control data happens on a periodical basis, in order to update the soft state kept by each node about the status of remote buffers: the loss of few messages is in no way critical to system operation. The length of UDP control messages mainly depends on the size of the buffer Trading Window, and is typically less than 100 bytes. The membership HELLO messages, in the order of the few tens of bytes, are also sent over UDP. In the original plan, all network messages were meant to be encoded in the Bencode format by Bram Cohen, but lack of time and resources has imposed the provisional use of a less efficient text-based protocol.

Bandwidth Allocation and Rate of Control Messages In a practical context, the way bandwidth allocation is performed is critical as it often determines the real efficiency of a data distribution application. While the algorithms specify which data chunks have to be sent to which neighbor, an application is supposed to perform the data transfers using a network whose behavior is unpredictable: the application must carefully manage sockets and buffers, especially when sending out data, and be responsive to unexpected events by using timeouts and triggers, to mitigate the negative influence of churn and congestion.

As we described above, the PULSE node opens a small number of connections to its neighbors for data-exchange purposes. In practice, after the scheduling algorithm at the sender peer assigns a chunk to a certain neighbor, the chunk enters a per-neighbor sending queue. The presence of the queue decouples the scheduling decision from the transmission of the chunk on the socket interface. To avoid starvation, the queue can accept only a small amount of chunks (currently two), and a new space is freed only after the amount of data in the queue drops under a threshold. This simple mechanism allows to avoid both inefficient link use, as a fixed amount of data can always be sent without having to wait for a new scheduling round, and excessive queuing of chunks on slow connections.

One of the drawbacks of data-driven systems is the need for a steady control traffic to support data exchange. To limit the rate at which messages are generated between any two peers, we introduce a self-clocking mechanism based on data exchange. If no data are being transmitted or requested, RED messages are sent with a minimum fixed rate: this happens for the majority of peers that currently are in the RED knowledge list. The “active” peers (nodes with which an exchange is in progress) have a message rate that increases with the rate of data exchange and is upper-bounded by the stream chunk rate. Also, in order to reduce the number of active neighbors, the chunk request algorithm is slightly biased so that chunks are preferentially requested from peers which are currently active.

Chapter 4

Understanding the Behavior of PULSE

The previous chapter presented the PULSE protocol and algorithms. The rest of this thesis will be devoted to understanding how and why PULSE works, evaluating its performance and scalability, and - in general - validating whether the system architecture we propose is able to meet our stated goal, that is supporting a large-scale live streaming application in heterogeneous and dynamic environments.

In many respects, the biggest challenge about PULSE is actually *understanding its behavior*. It is indeed non-trivial to predict the global properties of a data-driven system from the simple description of the algorithms that are executed by its individual nodes. We were unable to provide a theoretical model of PULSE, as its algorithms are based on pairwise incentives coupled to a feedback response from the data distribution process: the presence of a dynamic component as the fundamental feature of our system has forced us to turn to alternative forms of evaluation, namely simulation, emulation, and limited deployment on the Internet.

This chapter reviews the various techniques available today to understand data-driven systems. Section 4.1 is devoted to theoretical modeling of static data-driven architectures. Section 4.2 describes the latest models for incentive-based streaming systems. Section 4.3 introduces simulation tools and techniques, while Section 4.4 examines the challenges and the advantages of performing an evaluation using emulation techniques. Finally, Section 4.5 gauges the feasibility of large-scale system deployments and the potential of measurement studies.

4.1 Modeling Static Systems

The literature on the theoretical analysis of peer-to-peer streaming systems has kept growing with a fast pace during the last six years. Most of the early work focused on the study of overlay-driven and structured systems, which posed no particular challenges due to the geometrical properties offered by their explicitly-built overlays. Data-driven unstructured systems, on the other hand, have not been the subject of a comprehensive theoretical study until very recently: the available techniques to model these systems in realistic conditions are not yet mature and still constitute a vibrant research field. While the first models for data-driven systems were introduced to study practical bulk data distribution systems such as BitTorrent [120][90], they have been quickly extended to the domain of live streaming.

Fluid Models Fluid models, an application of markovian queuing theory, have been used extensively in the past to model the complex dynamics of computer networks at the packet level. They have been recently applied to content distribution networks and to P2P live streaming [66], providing interesting results on the upper-bounds of the streaming rate under simple bandwidth distributions and node churn patterns. By their very abstract nature, however, fluid models are not appropriate to describe data-driven systems, as the internal state of the nodes (buffer contents, local knowledge) is never taken into account. The absence of a more precise characterization of the individual nodes makes it impossible to model the algorithms for peer and chunk selection and the underlying network topology.

Packet-Level Models Another approach to fluid modeling [72] introduces a packet-level markovian model over an edge-capacitated network graph, which is then extended to the node-capacitated case. This model allows to introduce and evaluate simple node selection and chunk selection policies: among other insightful results, the paper evaluates and proves the optimality (limited to complete network graphs) of a combined random-useful packet selection with a most-deprived neighbor selection. Unfortunately, the extension of this method to generic fixed graph topologies (and worse, variable topologies) is an open question. Also, the peer selection strategies that can be currently modeled are rather simple and cannot rely on the internal state of the nodes.

Gossip Models Gossip-based modeling uses a randomized approach to chunk distribution in order to obtain probabilistic bounds on the performance of peer selection and chunk selection schemes. In [95] several simple protocols that do not require data reconciliation between peers are presented, and their efficiency is proved to be comparable to fixed tree overlays. To the best of our knowledge, gossip models haven't been evaluated yet in scenarios with heterogeneous node capacity, peer churn, or when nodes do not spontaneously cooperate.

4.2 Analyzing Incentive-Based Systems

Most live streaming systems were designed to operate in cooperative environments where node contribute either homogeneously [21] or as much as they can [65][27]. When heterogeneous node capacities are allowed, few streaming systems introduce techniques that aim to increase the *social welfare* of the system [28], sometimes giving advantages to nodes that contribute more. The "reward" to better contributors can be in terms of locally increased media quality, e.g. when nodes are attributed a higher number of stream descriptions [107], or lower reception latency, e.g. when more resourceful contributors preempt less resourceful nodes and switch their positions

[98]: the system-wide effects often include improvements in the system-wide performance and efficiency of data distribution. Obviously, these techniques depend on the willingness of all the peers to abide to the protocol rules: typical ways of cheating include malicious nodes behaving as if they were good contributors or ignoring preemption requests.

The adoption of incentive mechanisms can allow live streaming systems to work in non-cooperative environments: incentives provide a local source of information about node contributions, which is usually leveraged to give back to the peers that contribute and, conversely, to retaliate against the nodes that do not contribute sufficiently.

4.2.1 Intrinsic Incentives

In the general context of data distribution, finding a good internal balance between fairness and altruism is hard. Studies about BitTorrent have shown the existence of a fundamental trade-off between performance and fairness [42]. Moreover, the presence of even small amounts of altruism, while boosting performance, opens the way to easy exploitation by greedy nodes [68]. In the context of live streaming, the presence of timing constraints further complicates the issue, as the incentive mechanism should not diminish the efficiency of data replication [31].

Pairwise Trust Pairwise trust incentives rely on direct assessments by individual peers about the capabilities of other nodes. Live streaming applications are particularly suited by trust-based incentives since *(i)* all peers share interest for the same data at the same time and *(ii)* nodes have to interact repeatedly over time as long as streaming is being performed. Examples of current live streaming systems that integrate trust metrics use it to detect and retaliate against freeloaders [46] or to provide to better contributors a higher number of backup paths to mitigate the effects of churn [97]. As far as we are aware, the effects of these trust policies on the global behavior of large-scale streaming networks have not been evaluated.

Game Theory The use of game-theoretical incentives in peer-to-peer networks has been introduced quite recently [33] as a way of ensuring that the peers, modeled as rational agents, have a reason to voluntarily provide their resources to the system [11]. Game theory (GT) provides a framework to describe and evaluate P2P systems as *non-cooperative games*: nodes are modeled as players, game decisions are mapped to resource contributions, and the game consists in having nodes play against the rest of the system, striving to maximize their own *utility function*. The study of *Nash equilibria*, defined as the global states of the system in which no player can increase its own utility by changing its current strategy, provides an analytic tool to determine the existence of stable (pure) trading strategies of an iterated game.

Recently, [19] proved that, eventually, a system with differential incentives converges to a Nash equilibrium. In [108], several pure strategies including freeriding are introduced and their effects evaluated. Games such as the “rate game” [90] and the “network formation game” [75] have been used to analyze the properties of the BitTorrent overlay, showing the existence of a Nash equilibrium for the cases of homogeneous and heterogeneous node upload distribution. However, the incentives implemented in practical distributed systems are often analytically intractable as they depend on a too large number of real-world parameters and constraints[1]. The limits of classic GT as a modeling approach for real-world incentives arise from the very abstract nature of game-theoretical models. Everything, from the system-wide utility function to the acceptable player strategies, must be arbitrarily set: while the *qualitative description* of real systems may be very accurate for specific choices of strategies and parameters, the *predictive value* of game-theoretical models is quite low.

4.2.2 External Incentives

Systems that adopt external incentives rely on *separate protocols and facilities* to help nodes decide about their interactions with other peers. Because of their independent nature, extrinsic incentives are usually separate subsystems with an extremely limited interface to the internal state of the main streaming application. Incentive subsystems tend to manipulate values of system-wide variables (such as a node's “reputation”, “wealth” or performing micro-payments and currency exchanges [114][74]) based on information from the local and remote nodes. The use of reputation-based incentives has been advocated for instance in [89], however without a published evaluation. An analysis of systems based on extrinsic incentives lies out of the scope of this thesis.

4.3 Simulating a Distributed System

As the current theoretical models does not offer the support we need to evaluate the properties of our system and to predict its scalability, we had to attack the problem in an empiric way. Simulation, as usual, comes to the rescue when we need to model a phenomenon that is not sufficiently understood. The main strength - and drawback, at the same time - of simulation is that it relies on a simplified model of reality, where the metrics that better characterize the system can be easily observed and measured. The predictive value of a simulation is lower than mathematically-proved theoretical results, but nonetheless sufficient to anticipate the behavior of a system over a wide range of parameters and border conditions. On the other hand, simulation results have a much lower generality and cannot be easily exported to different contexts. Simulation, while unable to perfectly reproduce the dynamics of real-world systems, can provide useful insights with limited requirement of human and computational resources.

[1] A relatively new branch of GT, called *evolutionary* GT, studies the spontaneous emergence of dominant strategies when each node in the population is free to evolve its initial strategy, e.g. by copying the strategy of more successful peers or introducing random mutations: the goal of evolutionary GT is to develop empirical non-deterministic strategies that can induce a desired global behavior [53].

4.3.1 Background

A number of simulators is available to simulate the behavior of computer networks and distributed applications: as a thorough simulation performed at the lowest level (i.e. bytes on a physical medium) would be very much time and resource-consuming, each simulator usually concentrates on modeling a specific facet of the interesting phenomenon.

The most accurate packet-level simulator in common use is *ns2* [15]: its primary goals are providing a faithful representation of TCP protocol behavior, interaction between routing and topology, and effects of heterogeneous physical media on transport protocol performance. Other simulators of this type but with somewhat more limited purposes are REAL [64], showing a specific interest in TCP and packet scheduling (apparently discontinued since 1997) and SSFnet [76], focused on TCP/UDP and routing protocol performance.

Performing an accurate simulation, however, has its costs: these costs translate in high requirements of processing power, memory bandwidth, and RAM capacity by the hardware on which simulations run, and in long waiting times for the people performing the simulation. An approach such as the one used by ns2 can be viable when studying the performance of data exchange between few nodes across a network, but shows clear scalability problems when analyzing the performance of a distributed protocol, with thousands of nodes exchanging data at the same time. Nonetheless, the level of accuracy offered by this approach could be required to successfully model a specific protocol: simulators like P2PLP [54] act as a front-end to a network simulator, transforming a high-level description of the distributed algorithms into a series of network events, which are then passed to ns2 for execution and whose feedback is taken into account in the subsequent phases of the simulation.

When highest accuracy is not fundamental to the application being analyzed, it is possible to lower the requirements of the simulation by adopting a simpler network model: this typically leads to a trade-off between upper limits in simulation scale and accuracy of the results. Different applications can withstand different amounts of simplification, so the system designers have to use their own judgment when choosing an existing simulator:

- Few simulators strive to be as generic as possible: they use an event-driven main loop and just simplify the transport protocol algorithms to improve their speed over ns2. An example of such a simulator is GPS, the Generic P2P Simulator [119], which claims a good accuracy along with reasonable speed. GPS exposes all network entities as objects, giving great flexibility to the implementer of new application protocols: the example protocol provided by the author models the BitTorrent protocol, which is a quite demanding application in terms of information required by each peer to operate correctly. On the other hand, the claims of accuracy are not supported by a thorough analysis and validation on a significant scale.

- Some simulators give an option to replace the classical *event-driven* main loop, which is used by most network simulators, with a simpler *timeslot-driven* structure. This choice usually means that the role of underlying topology and network latency is eliminated from the model, but still leaves the freedom to experiment with bandwidth allocation and peer selection policies. For example, PeerSim [60] is a generic simulator that allows to experiment with structured and unstructured distributed protocols; however, it is not clear if the limited set of parameters that are made accessible to the nodes allows to faithfully model the complex internal state and local behavior of unstructured distributed algorithms.

- Finally, most simulators purposefully restrict their applicative scope to be specific to a limited subset of system architectures: this usually implies simplifications both in the network model (time-slots) and in the range of actions available to simulated nodes.

Looking at the simulators that were available at the time we performed our analysis (January 2005), it was unclear whether they would be suitable to our task. The main concern we had was about whether the feature set they provided to implement the behavior of a peer was sufficient to support the complex internal state required by the PULSE algorithms. This concern, along with the fact that we did not know yet what would be the appropriate granularity for a realistic yet fast simulation, pushed us to choose the last option, writing a purpose-built simulator.

4.3.2 Pulsim - The PULSE Simulator

Simulating an experimental phenomenon requires that the people doing the modeling first get acquainted with the effects of the phenomenon, experiment a little, play with its variables, etc., to gain a first insight on what is its ordinary behavior and on what standard modeling technique would work best. On the other hand, modeling an *unknown system* - for which no experimental data is available yet - is *much harder*, as the researcher typically has no idea of a) what the normal behavior of the system is, b) on what timescale this typical behavior is best visible, and c) whether the system itself could actually exist as an experimental phenomenon. For these reasons, writing a simulator for such a system is necessarily a recursive trial-and-error process, based both on intuition and conjectures from partial results.

In the beginning, we approached the task of simulating PULSE from a realistic point of view: we conjectured that the only way to properly model the complex feedback-driven algorithms that each node implements was to take into account in the simulation all the variables that appear in the real system. This basically meant to emulate as deeply as possible the network (link latency) and transport (delay of data transfers), over which a gossip membership protocol would run (with BLUE knowledge messages appropriately delayed), that would in turn provide the information required by PULSE to function.

We initially adopted an event-driven approach, with a minimum step granularity of 10^{-4} sec to ensure that the asynchronous nature of the system would be preserved (small probability of concurrent events happening). The input accepted by the program included the node bandwidth distribution and a full pairwise latency matrix. This approach was found not to be viable because of the extreme computational complexity of the simulation, and the difficulty debugging the protocol activity at so many levels. However, this initial experience suggested that the most relevant factors in the behavior of the network would be the delays due to data transfers, mainly because of the size of the chunks (tens of KB's), and the propagation process of knowledge about data chunks among the peers. The dominant factor shouldn't have been the fact that data was traveling back and forth over a simulated network, but that nodes had to *take decisions* based on their local view of the system

For these reasons, we chose a coarser granularity for our final design, switching at the same time to a timeslot-based approach: this emphasized the role of node buffers, peer choices, and local knowledge about the data, while still allowing control on the most important parameter relative to users of the system, the node upload bandwidth. Also, the emulation of the BLUE gossip subsystem was abstracted out with an oracle-like global mechanism: the rationale was that, since the role of blue messages is significant only in the earliest phases of a node lifetime - when there is no other source of information about the rest of the system - its importance significantly decreases near steady state. Moreover, using an oracle assures that no node will find itself 'isolated knowledge-wise' from the rest of the system due to an unlucky turn of events, eliminating this negative interference from the simulation outcomes.

There are drawbacks to these choices: first of all, network locality had to disappear from the system. This was a loss that was difficult to accept, since propagation times of the messages appear as a core parameter in the PULSE peer selection algorithms. Still, its relevance is arguably much lower than the role of bandwidth in determining the choices of peers, as the tit-for-tat mechanism only consider how many chunks have been obtained from a node[2]. Then, the presence of an oracle has a side effect in the fact that it exposes a partial subset of private node data to the whole node population: one consequence of this is the over-inflation of the local view at each peer. We verified that the ability to contact any other connected node can dilute the effectiveness of the peer selection algorithms, especially in the initial phases when no past knowledge about the performance of other nodes exist.

The full details of PulSIM, along with a more accurate description of the motivations, limits, and strengths of its simulation model, are presented in Chapter 6.

4.4 Large-Scale Emulation

Emulation techniques are very powerful, as they can give reliable and reproducible empirical results about the real behavior of algorithms in distributed systems. Emulation allows to deploy a medium- to large-scale P2P network while retaining the full control of all its operating parameters. Also, the emulated behavior of an algorithm implementation can take into account important practical factors which often are not easily tractable by theoretical analysis, as the

[2]Network latency has two second-order effects on the bandwidth that is exchanged between peers: first, as control messages take longer to propagate, if proper request pipelining is not implemented the available bandwidth could be exploited in a less efficient way; second, as control messages that are in-flight for a longer time contain older information about the remote node's buffer, this could reduce the probability that the remote node may provide interesting chunks.

availability of knowledge at the nodes and the realistic allocation of bandwidth. Emulation of data-driven systems is a fundamental step to gain a reliable understanding, as peer interactions, data availability and bandwidth allocation can be modeled for the first time in a realistic way, and often allows to validate or invalidate the predicted behavior suggested by simulation models.

4.4.1 Implementing a Prototype Node

One of the first choices that must be made when starting a software project is which language and framework will better suit the needs of the project. In our case, a balance had to be found between rapidity of the development phase, usefulness of external libraries, platform-independence of the resulting software, and maintainability of the code. These reasons motivated our choice to use an object-oriented interpreted programming language and event-based network library: in hindsight, we believe our choice was right, as the prototype was completed in the expected timeframe and could exploit a range of readily available software components that accelerated the development and debugging phases.

The drawback of choosing a high-level programming language and complex libraries is the relative loss of control on the low-level operation of the software, especially on the way it manages network sockets and buffers. Implementations of applicative protocols are often hard to compare, as it is difficult to abstract the strict performance of the algorithms from the surrounding implementation details. In order to facilitate comparisons, sets of network libraries, such as MACEDON [94] (for C++), have been developed to make it possible to experiment with the various algorithms on a common ground. However, while this approach has been adopted to compare tree-based systems (from NICE to Bullet to Splitstream), to the best of our knowledge no data-driven systems have been implemented using this framework yet. Extending this approach to data-driven systems could provide experimental data to support more extensive theoretical comparisons between data-driven and tree-based systems than the existing ones [71].

4.4.2 Emulating a Large-Scale System

Compared to simulation, emulation helps assess how the implementation of a set of algorithms behaves. This means that the actual application code - executable, self-contained, and capable to interface to real networks - is operated under controlled network conditions. Large-scale emulation demands powerful computational resources, proportional to the target system size, but has the advantage of *not requiring a real network infrastructure:* tools such as Dummynet [92] and Modelnet [111] help simulate in a local setting a configurable internetwork substrate with realistic pairwise latencies and link bottlenecks. Simulating the network topologies while leveraging the computational resources of a large scale research grid, such as Emulab [118] or Grid'5000 [3], allows to extend the emulation approach to an order of magnitude of thousands and tens of thousands nodes, which approaches pretty well the expected scale of an initial Internet deployment.

Since the conditions in which emulation is performed are both controlled and specified *a priori* by the researcher, emulation does not answer all the questions about how the emulated system will fare in an uncontrolled real environment. While the network can be configured at will and factors such as cross-traffic and geographic diversity of the nodes can be added to the model with relative ease, the fact that all the conditions are kept under control means that the effects of unpredictable real-world phenomena, such as random node failures user dynamics, could still have an unexpected impact on the behavior of the application.

4.4.3 PlanetLab

To better approximate realistic operating conditions, Internet-based testbeds such as the Planetlab [5] can provide a practical way to experiment a limited-scale deployment in the order of few hundreds of simultaneous nodes. While the traffic generated by the deployed application travels over the Internet, the experimenter has little knowledge about the state of the network and of the computers, which may be used at the same time for an unknown number of other experiments: this makes PlanetLab results non-reproducible and quite ill-suited as a basis for performance evaluation and comparison. However, the fact itself that a system can operate in these condition is a partial confirmation of the practical viability of a networking application.

Our experience with PlanetLab has suffered from well-known shortcomings due to the unreliability of many PlanetLab nodes and their high average system load: while most measurement and data distribution experiments may demand little processing resources and be able to run without any adverse effect from load, in our experiments with PULSE we have observed that CPU load was in many cases so high to prevent the timely execution of expected protocol actions, leading to problems ranging from time-outs to completely erratic software behavior. Finally, recent studies argue that the placement of PlanetLab nodes is not representative of the typical connectivity of Internet hosts, as they are mostly placed along the well-provisioned international research and educational backbone, and that their network diversity is lower than the average diversity in the "commercial" Internet [13]. While this limitation mainly affects measurement studies, it may also have an impact on the behavior of application-layer protocols.

4.5 Deploying on the Internet

A definitive answer about the practical behavior and viability of a large-scale network can only be found by actually deploying a large-scale system and measuring its performance. The challenge of deploying a live streaming system to users world-wide is a quite impressive one[3]: the development effort required to test the software, fix its bugs, implement new features, monitor the health of the system in real time, and recursively improve the software and algorithms to follow the growth of the system, all of this requires the combined effort of either an industrial team or of a small development community. Then, to attract enough attention and reach a large

[3]Gale Huang, PPLive Software Architect, Keynote at the 2nd SIGCOMM P2P-TV Workshop, 2007

audience a marketing strategy is required, as rarely word-of-mouth alone is sufficient to immediately reach a large user base: this task becomes increasingly difficult as more and more players enter the same "technological niche". Clearly, such an endeavor requires a lot of effort, and is unlikely to reach the system scale and audience size initially sought.

4.5.1 The Potential of Measurement Studies

Once a system has reached a sufficiently large deployment status, it suddenly becomes an interesting object for measurement studies. The earliest large-scale commercial live streaming systems have been steadily gaining popularity since 2005, and today have reached impressive sizes. Starting in 2006, measurement studies about SOPCast and PPLive, two of the largest live streaming systems, have begun to appear. These studies have provided performance results from the perspective of the end-user [9], "black-box" analyses of the external protocol behavior [102], and led to the partial reverse-engineering of the PPLive protocol, which allowed the implementation of a crawler capable of capturing the full-scale dynamics of popular live streaming sessions [55][56].

Measuring the behavior of a working system and the patterns of user activity provides extremely valuable insights on the nature of the problem itself and on the practical challenges that system designers have to take into account. Measurement studies have a high practical value as they allow to directly compare the efficiency and scalability of real-world systems in similar environments: while they are not repeatable and do not provide context information on the state of the underlying network during system operation, they give the most useful information about the performance of the application and on the overhead of the system under a real workload[4].

[4]Another drawback is that the behavior observed on specific applications is often hard to generalize to different applicative contexts. The results in [18] suggest that even minor non-structural details - such as GUI modifications, addition of new "cosmetic" features, etc. - often have unexpected effects on the way users interact with an application.

Chapter 5

Metrics For Performance Evaluation

5.1 Introduction

This chapter focuses on two fundamental questions: how is it possible to measure and compare the performance of generic data-driven systems, and how is it possible to successfully describe and evaluate the behavior of the PULSE algorithms.

5.1.1 Dealing with a Data-driven System

In Chapter 2 we described in-depth the differences between data-driven and structured overlays: in synthesis, tree-based systems have the advantage of explicitly defining a single path for data distribution - which makes analysis easy and allows to straightforwardly introduce optimizations (latency, bandwidth) - while in mesh-based systems the data distribution trees are an indirect consequence of several mechanisms, such as peer selection and chunk scheduling - which loosen the control on the path taken by data over the network and hinder deterministic optimization techniques.

From an analytic point of view, data-driven systems are harder to describe and evaluate than classic structured systems. The reasons are the following:

1. Data do not follow a constant path over the mesh, nor the trajectory of a data chunk in the system is predictable, even given the complete knowledge of the state of all the peers in the network.

2. Local mechanisms for data reconciliation are needed to avoid unnecessary data duplication. Data reconciliation requires the exchange of control information about data chunks carried by each node. Tracking the knowledge of the rest of the system and the decisions made at each node is quite challenging, when looking from a global perspective.

3. Connections between nodes can carry either control information only or both data and control information. The existence of a connection between two nodes does not imply that data are being steadily exchanged in either direction.

4. The mesh is frequently changing over time, as the connections that constitute it are renegotiated locally among the peers.

5.1.2 "Mesh Overlays" and Performance Metrics

In the general data-driven case, we have seen that the classical definition of "overlay" is stretched to its limits. "Connections" in an unstructured mesh overlay are both functionally and semantically different from those in systems with structured, tree-based overlays. Also, the resulting performance of data distribution is not completely determined by the placement of nodes and by their resources, but are also influenced by the availability of useful data chunks and up-to-date knowledge about neighbor nodes. Lacking geometric structure and average per-connection service expectation, a data-driven overlay does not offer any property that could help predicting its global performance.

While many structured overlay multicast systems can be (and have thoroughly been) analyzed starting from topological considerations, data-driven systems have so far proved impervious to this kind of analysis. As the basic assumptions that are commonly made when dealing with traditional, structured systems (such as single or multiple distribution trees, etc.) do not hold for data-driven systems, performing a satisfactory *a priori* analysis is very hard and still constitutes an open field of research. We will have to adopt an empiric approach to performance evaluation in order to deal with data-driven systems in general (and PULSE in particular).

PULSE exhibits all the properties of a data-driven system, and - additionally - its mesh topology evolves quickly over time and is influenced by feedback from the data exchange activity between peers. There are no explicit rules or constraints for building, repairing, and optimizing the overlay, only the *peer* and *chunk selection* algorithms running on the individual nodes (which we described in Chapter 3). These algorithms constantly strive to establish efficient partnerships between peers, and to choose the chunks that need to be redistributed. However, their effectiveness in a specific scenario (bandwidth availability, bandwidth distribution, latency distribution, etc.) cannot be evaluated with the classic metrics used to describe structured overlay topologies.

5.1.3 Outlook

In this chapter, we discuss the issues we encountered in our attempt to study the behavior of data-driven systems and the solutions we devised to perform our analysis. In Section 5.2, we try to adapt some of the traditional performance metrics used to study live streaming applications to make them suitable for generic data-driven systems. Section 5.3 introduces additional metrics to evaluate the specific behavior of PULSE and to understand what happens inside a running system. All these different metrics will be the main tool at our disposal to understand how PULSE fares in different scenarios, how it responds to external perturbations (such as node arrivals, departures, variations in the available bandwidth, etc.) and how some slight parametric changes of the core algorithms affect its global performance. Finally, Section 5.4 concludes the chapter.

5.2 Performance Metrics for Data-driven Systems

As we stated above, all the classical overlay performance metrics that have been used to evaluate structured systems cannot be applied to a data-driven system, since they would require a fixed and predictable geometrical characterization of the overlay. Even the most basic metrics, such as data reception delay and data loss rate, cannot be used as they are, as the distribution of data chunks does not follow a sequential order. We clearly need adequate empirical tools to describe generic data-driven systems and analyze them in a comparable way to structured overlays.

5.2.1 Data Reception Delay at the Nodes

One of the requirements of a live streaming application is that *the stream data must be received in order* before the media can be played. We can reduce this constraint to the concept of *completeness* of the stream data at playout time: a data loss event involves pieces of data missing their playout deadline, even if the missing data could be eventually retrieved afterward.

Remarks about Data Loss In general, the architecture of the streaming system determines its data reception requirements. For instance, the earliest systems based on single-tree structured overlays considered data as a continuous bit stream that was received from the parent node. Buffers at the nodes were small and dimensioned so that they could absorb the delay jitter on the connection with the parent.

In these systems, data loss is possible in case a serving node has a congested uplink or if an ancestor node leaves the system: losses due to disconnections can usually be detected when no data are received during a certain time interval (this timeout is usually set to a value around $\mathbb{E}(delay) + k \cdot \sigma(delay)$, where the constant k allows to trade between detection sensitivity and maximum response time), while losses due to congestion might be detected if the data is found to be damaged by the application or when an unexpected datagram (i.e. future, out of order) is received.

The reaction of a node to data loss is system-specific and depends on the size of the playout buffer, on the underlying transport protocol used by the application, and on the presence of loss recovery mechanisms. In interactive and "almost real time" systems, nodes often do not try to recover the loss but - depending on its entity - either compensate it using redundant media encoding, try to hide the resulting video/audio artifacts, or display the media with degraded quality. Some systems also react to repeated loss events by dynamically decreasing the rate at which the media is encoded, either at the source or at the parents of those nodes which are experiencing losses (e.g. down-sampling, transcoding [25]).

The definition of data loss is less clear-cut for data-driven systems, as the stream is no longer treated just as a sequence of bits but is now a sequence of chunks. Each chunk has a sequence number, and mechanisms are in place to ensure that its contents are complete (e.g. reliable transport protocols, application-level checksums). In data-driven systems, chunks are normally retrieved out of order. The definition of loss has thus to take into account the deadline by which the chunk has to be retrieved, e.g. the time left before the chunk playout is scheduled. Chunks that are received after that deadline, in addition to constituting loss events, also have the adverse effect of wasting bandwidth.

Depending on the average playout latency targeted by the system, different mechanisms to avoid losses have been devised. These mechanisms can range from appropriate chunk scheduling techniques [122] to more complex buffer management algorithms, as we currently use in PULSE. Redundant media encoding techniques [93][78] can again be used, trading bandwidth usage and CPU overhead for an improved resilience against chunk loss. On the other hand, dynamically reducing media quality in data-driven systems would be impractical, as the source has no way to control the propagation of the data inside the system.

Continuity Index Two design options have become especially popular among the architects of practical data-driven systems. The simplest option is based around a data buffer that outputs data to the application at a constant rate. The chunks that do not meet the playout deadline are lost altogether, but the playout advances with a constant pace, even if the played media becomes severely corrupted. A typical metric for data loss in this context is the *Continuity Index* (CI), which is defined as the ratio between the number of chunks received on time and the total number of chunks [122]. In formula, for a system with N nodes over the time interval T:

$$CI = \frac{1}{N}\sum_{i}^{N} \frac{<\text{number of chunks received by } P_i>}{RT}$$

This metric is based on two silent assumptions: that the amount of disruption introduced by chunk losses is roughly proportional to the number of missed chunks, and that any distribution of a same amount of loss events over time roughly has a similar impact on the perceived quality.

The second option is based on a buffer that can selectively output data to the application, waiting for enough chunks to be collected prior to playout in the hope that loss events can be avoided altogether. With this approach, the playback may be frozen until an adequate amount of contiguous chunks has been received: in this case, the playout disruption can be limited to the fixed amount of losses the buffer is expected to tolerate. The main drawback of this mechanism is that every freeze increases the playout delay without any explicit upper bound, as long as the media playout is always performed at its nominal rate.

While the CI metric could be extended in some cases to this second type of systems (depending on the way the buffer tolerance is implemented) we must argue that the CI alone is not adequate to describe the 'reception quality' of a generic data-driven system. For instance in PULSE, since we combine a loss-tolerant selective buffer with an appropriate FEC encoding of the stream to guarantee that the playout quality will never suffer in case of chunk loss, the fact of receiving less than 100% of the chunks does not imply a degradation in the quality perceived by the user. The introduction of FEC allows in our case to seamlessly absorb $S = W - K$ losses over each window of W chunks. More than S losses, on the other hand, can have a strong non-linear effect on playback quality and may render the whole window of chunks unrecoverable[1]: PULSE responds by freezing the playback up to a maximum delay value, and then resetting the buffer – completely losing a whole segment of stream, rather than reproducing it with reduced quality.

The inadequacy of the CI to describe the performance of PULSE encouraged us to look for an alternative and complementary set of metrics that can be extended to all data-driven systems. These metrics are the playout and average chunk reception delays, which will be the subject of the following pages.

Instantaneous and Average Node Lag Using the reception delay, or *node lag* (Chapter 3), to describe data-driven systems allows us to overcome the limits of traditional metrics that rely on geometrical properties of the overlay. Whereas the position of a node in a structured overlay was clearly defined in terms of either number of hops or data reception delay from the source, in an unstructured context we can define the *position of a node* in terms of node lag.

A drawback of this metric is that the information about the actual path followed by the individual chunks and about the provider nodes is not taken into account, as we do not rely on any topological information but only focus on data reception in order to compute it. However, node lag still conveys a stable and meaningful description of the average "length" of the path that chunks follow to reach a node from the source: at steady state, its value synthesizes the average number of hops that chunks traverse, along with the average delay properties of each hop.

For these reasons, we will use node lag as the primary metric to represent both the position of a node inside the system and the large-scale evolution of a dynamic overlay.

Large-scale Evolution of Node Lag The value of T_B is the result of an averaging process over a short time span: it is thus sensitive to the long and medium-term variations of the node lag, such as those occurring when a nodes waits for specific chunks before its window can slide, or when a node starts receiving data from a bandwidth-rich neighbor. As we explained above, we can trace the dynamics the data reception process of a node just by looking at the variations over time of its average node lag.

Lag traces from individual nodes, however, just describe the evolution of individual peers inside the system: if taken alone, they do not allow a full understanding of the reasons behind the observed global behavior. But if we manage to extend this analysis to the entire node population

[1]The magnitude of the playout disruption depends on the way the coefficients for the Reed-Solomon coding are chosen and on which data chunks were lost, in addition to the more obvious factors such as chunk size and media format/codec.

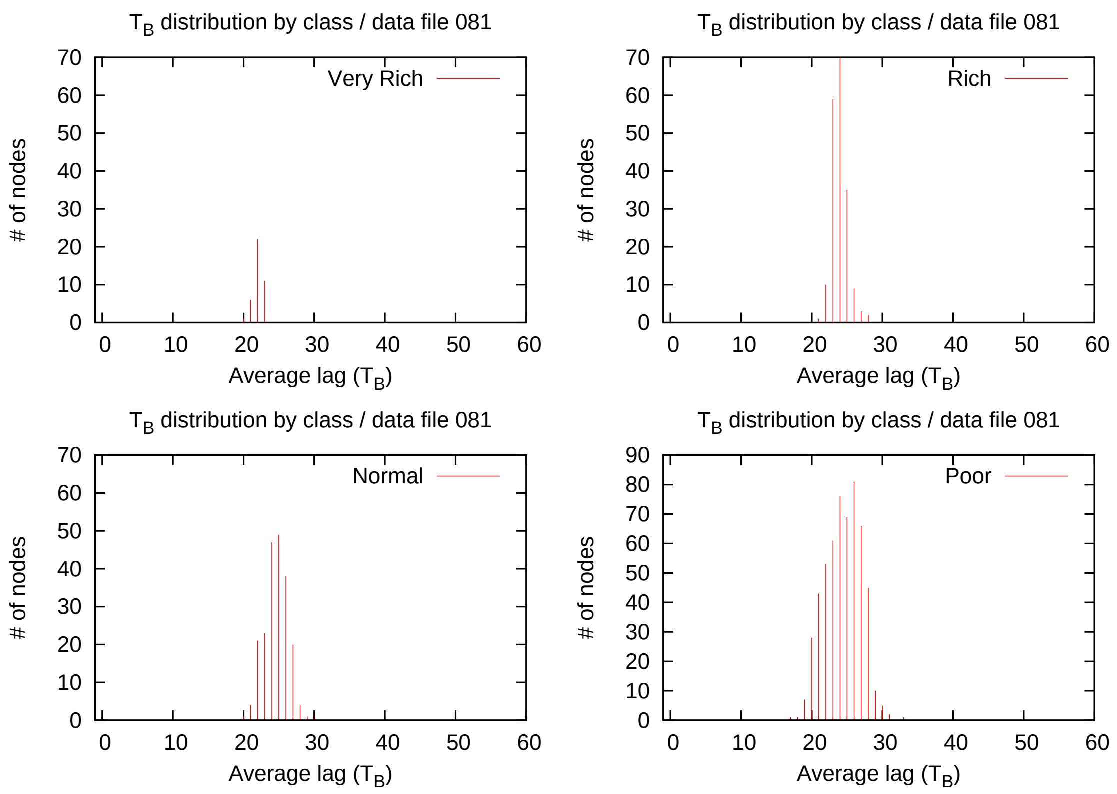


Figure 5.1: Histogram of the Number of Nodes per Lag Value by Class ($t = 81$)

and to appropriately correlate the individual traces, it then becomes possible to investigate in depth the reasons for the observed node behavior patterns.

Without loss of generality, we suppose that the node population is divided into a number of subsets with different properties. For instance, we may partition the nodes into several *bandwidth classes* according to the amount of upload bandwidth that is available to each node[2]. We can then draw an histogram representing the number of nodes from the same class that have the same value of T_B at a given time instant (e.g. as in Figure 5.1). By comparing the lag distribution of nodes from the various classes visually, we can already get a first insight about the global behavior of the system. It is easy to see, for instance, what are the smallest and largest values of node lag in the system, and how the lag is distributed.

Also, we can aggregate the lag information by class to get a synthetic picture of the aggregate behavior of similar nodes. This kind of analysis can identify both the presence of widespread problems in the data exchanges, such as massive disconnections, and class-specific results in reception performances. The average instantaneous value of per-class T_B, along with its distribution around the mean value (captured by statistical estimators such as the unbiased sample

[2]Grouping the nodes by their available upload bandwidth will be especially interesting when studying PULSE. Since one of the fundamental aspects of the behavior of PULSE is the presence of correlation between the available outbound bandwidth at a node and its reception performances, it will be useful to evaluate separately and then compare the average aggregate behavior of each bandwidth class.

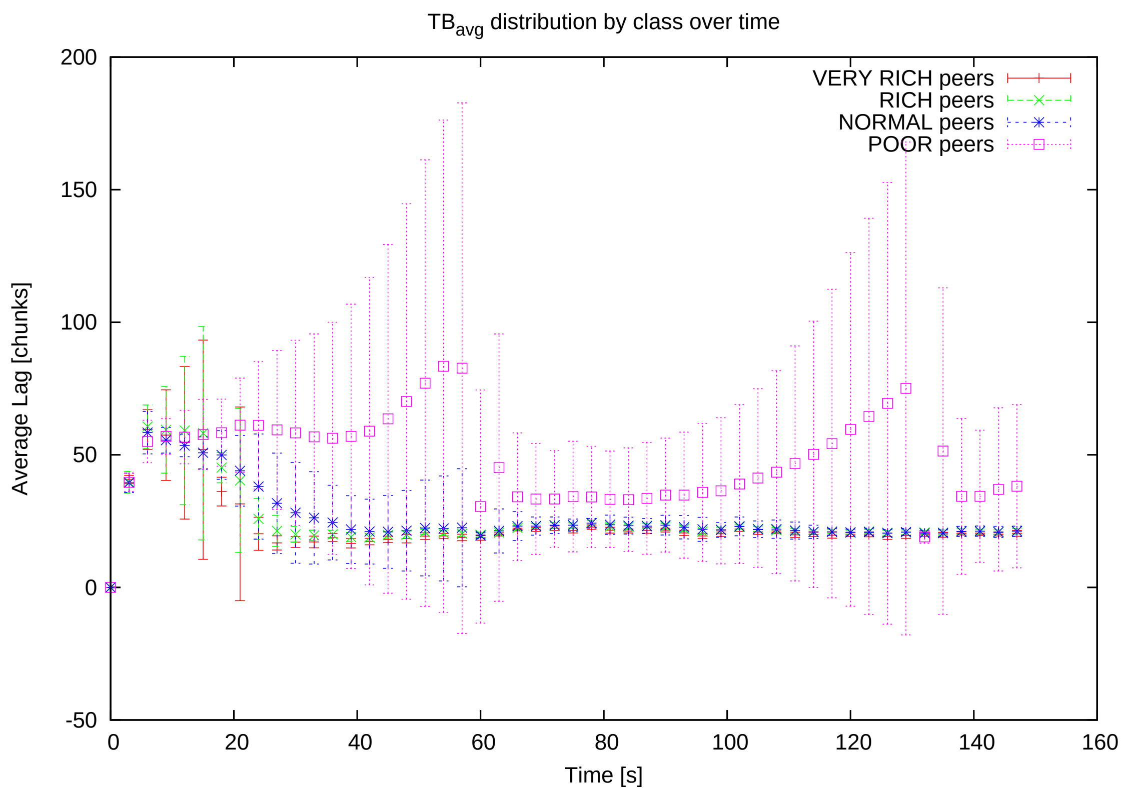


Figure 5.2: Plot of Average Class Lag and Lag Variance over Time

variance or appropriate percentiles) is an example of useful metric that can be easily computed based on node lag information. It becomes therefore easy to represent the statistical evolution of per-class node lag over the time domain: we can for instance represent the values of the average and variance of the per-class node lag over time, as shown in Figure 5.2.

Asymptotic Behavior of Node Lag Another interesting analysis we can perform using the node lag metric focuses on the scalability of a data driven system. In tree-based systems, scalability was more or less considered a fact - at least on paper[3] - since in the case of a balanced tree of fixed degree d the maximum delay needed for data to travel from the source to N nodes is $O(log_d N)$, as they must traverse at most $log_d N$ hops to get to the leaves at the bottom of the tree.

When dealing with an unstructured data driven system, however, understanding how it performs when it reaches larger and larger scales is a necessity. As the data distribution process does not follow predictable paths, it may be challenging to determine the relationship between the system performance at steady state and the size of the node population. Intuitively, as the number of

[3]Scalability in this context becomes a matter of using appropriate mechanisms to re-balance the tree in response to node churn, and to prevent nodes unable to serve a sufficient number of peers from occupy an internal position in the tree.

nodes in the system increases, the delay required for data to propagate from the source to all nodes will become higher and - consequently - the overall average node lag will also increase.

Average Node Lag vs. Chunk Reception Delay An interesting way to represent the status of the node buffers in the system is to look at it from the perspective of a data chunk. In a structured system, where the position in the overlay determines the data reception delay, the average delay for a given node is quite stable over time. In data-driven systems, as we explained before, chunk reception delay at a given node can fluctuate significantly over time, and out-of-order chunk reception is the norm. The average node reception delay, as defined above, describes the past and current performance of the node but does not give information on its future behavior. A new metric can be devised to estimate the timeliness of current data reception with respect to the average node lag, and again give a sort of prediction on the short-term variation of a node's lag.

For a chunk c_i, we can mark a point on a 2D graph when it is received by a peer P - with chunk lag $T^P_{c_i}$at time t - with coordinates $(\,T^P_B(t),\ T^P_{c_i}\,)$. The diagonal line $y = x$ represents the theoretical behavior of a structured overlay system, where the average reception delay is constant (i.e. $T^P_B = T^P_{c_i}$). Points above the diagonal represent nodes that received chunk c_i *later* than their average reception lag, while those below the diagonal represent nodes that got the chunk *before* their current node lag.

As we did before, we can perform the usual partition of the nodes into different classes. In Figure 5.3 we can see an example of plot comparing node lag with chunk lag for a population of 500 nodes in an early stage of a simulation, where the shape of each tick represents the bandwidth class of each node. We can draw many interesting insights from this kind of graph. For instance, we obtain:

- a snapshot of the position of the nodes by bandwidth class inside the system with respect to one chunk. This is equivalent to a simplified representation of the chunk's path inside the system (where the links followed by the chunk are not explicitly drawn).
- an estimation of the diversity of data buffers across the system, which is given by the horizontal width of the range of chunks that nodes are interested in requesting[4] (as seen on the x axis).
- the highest lag value with which nodes managed to receive the chunk[5], which corresponds to the maximum lifetime of a chunk in the system (as seen on the y axis).
- specific to PULSE: a glimpse on the state of the system. It may be converging, when nodes are spread in a comet-like fashion (as in Figure 5.3), stable, when all nodes are concentrated in a small area at a low lag value, or unstable, with nodes scattered around the plot in clusters of variable size.

[4]We must however remember that this representation is not a snapshot of the system's state at a particular instant, but rather a picture of the system as 'seen' by a data chunk.

[5]Obviously, not counting nodes that did not receive that particular chunk.

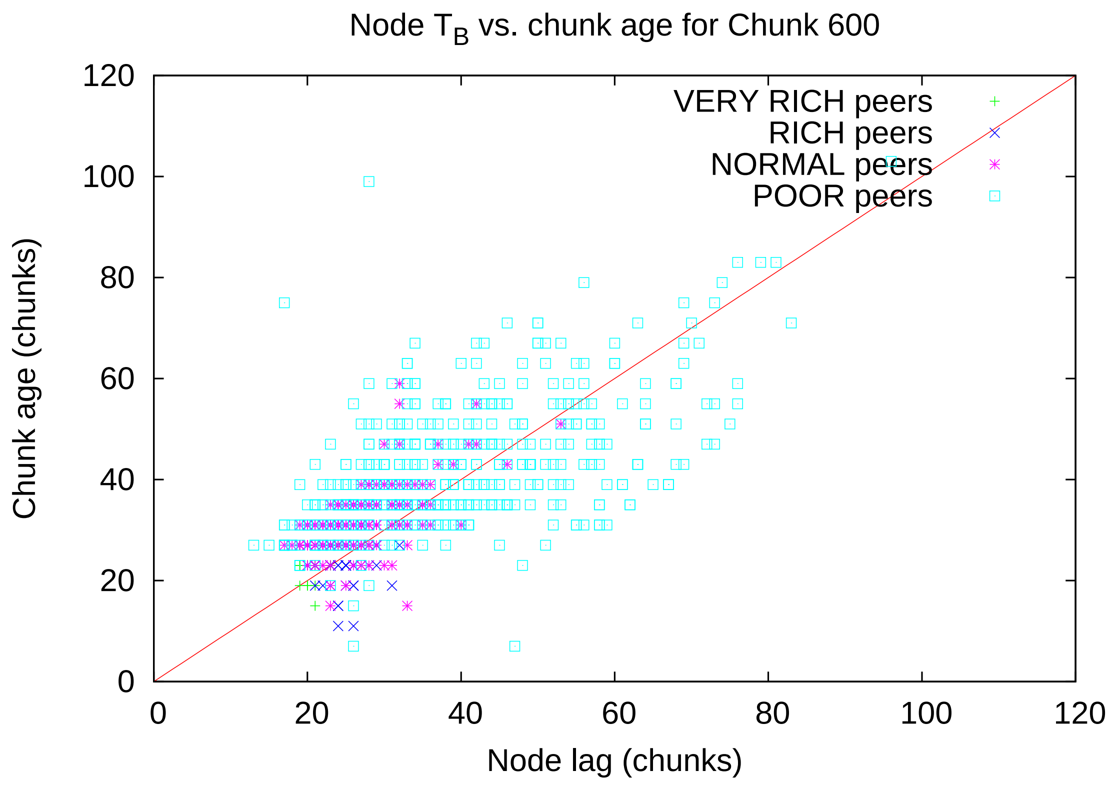


Figure 5.3: Average Node Lag vs. Chunk Reception Delay

5.2.2 Bottlenecks and Bandwidth Efficiency

The primary bottleneck of a streaming system is given by the upload bandwidth of the source: this bottleneck is a fundamental issue of all streaming systems for which there is no actual solution. Peer-to-peer systems alleviate this problem by allowing the users to contribute their resources, but the fact that the source is the only point from which data is introduced into the system still constitutes a bottleneck.

On Bandwidth Efficiency Let us consider a system with N nodes where each node has an upload capacity of U, and one source SRC with an upload capacity of U_{SRC}. The stream rate is SBR Kbps, which in the case of a chunk-based system means that R chunks of size $\frac{SBR}{R}$ Kbytes are generated each second. The total available upload capacity of the system at any time is $NU + U_{SRC}$: however, each node can use just a fraction $0 \leq \vartheta \leq 1$ of its available upload. We will define the *bandwidth efficiency* of a system as $\xi = \frac{1}{N}\sum_i^N \vartheta_i$. For instance, a node in a single tree-based system can only have an integer node degree $d = \lfloor \frac{U}{SBR} \rfloor$, which leads to a maximum efficiency (for an internal node) of $\xi = \frac{SBR\lfloor \frac{U}{SBR} \rfloor}{U}$, while leaves have $\xi = 0$. Nodes of a multiple-tree based system with M trees can reach a maximum efficiency $\xi = \frac{\frac{SBR}{M}\lfloor \frac{MU}{SBR} \rfloor}{U}$. Finally, data-driven systems can achieve $\xi = 1$ by using an optimal scheduling algorithm.

Bottlenecks in Structured and Unstructured Systems The connection bandwidth between a pair of connected peers $a \rightarrow b$ can be defined in a simple way as:

$$B_{a,b} = \min\left(\frac{D_b}{\text{in-degree}_b}, \frac{U_a}{\text{out-degree}_a} \right)$$

In structured acyclic overlays such as trees, the data reception rate of a node is upper-bounded by the stream bit rate, and its lower bound is determined by the smallest connection bandwidth on the overlay path to the data source, that is:

$$B_{SRC,n} = \min_{i->j \in path\{SRC,n\}} (SBR, B_{i,j})$$

Given the current scenario of bandwidth availability on the Internet, the scarce resource is located at the node uplink, which can be (in the case of ADSL links, the most popular commercial access technology) *two to twenty times* smaller than the downlink of the same node. For this reason, the main source of bottlenecks in structured overlays is upload bandwidth scarcity at an internal (non-leaf) node.

In data-driven overlays, while B is still defined for each connection, it becomes harder to define a bandwidth bottleneck on the data paths, since there are multiple ways of connecting the source to each receiver and since the distribution of chunks does not follow a sequential order. This situation is encompassed by the well-known Edmonds' theorem (1969), which defines the minimum theoretical streaming rate from a single source over an edge-capacitated overlay graph as *the sum of the edge capacities across the "minimum cut" of the graph* (the minimum cut is given by the partition of the graph for which the sum of edge bandwidths that cross the cut is minimal) [40].

Different chunks are commonly distributed over different spanning trees: depending on the chunk scheduling and node selection algorithms used in the system, the bandwidth efficiency can be noticeably improved and leads to an important reduction on the maximum chunk propagation delay. Karp *et al.* have proved in [62] that centralized scheduling algorithms exist that can approach the optimal propagation bound in the continuous-broadcast problem: while these algorithms cannot be applied to fully distributed systems, they suggest that the bandwidth efficiency of a system that allows data paths to change over time is potentially better.

However, the lack of an adequate link bandwidth (at least equal to the stream bitrate) on the system's "minimum s-cut" is not the *only* source of bottlenecks in a data-driven streaming system. The state of the data distribution process also determines which chunks a node *needs* and which ones it *is able to* retrieve: the actual bandwidth efficiency of the system could be further reduced because of an inefficient scheduling in the chunk distribution among the nodes. Therefore, the bandwidth efficiency of the system can be used as a metric to evaluate the functionality of a set of peer selection and chunk selection algorithms.

Bandwidth Bottlenecks vs. Content Bottlenecks We define *starvation* as the reception of a data rate lower than the bitrate of the stream by the receiver r. This phenomenon happens when $\sum_{i \in r's\, neighbors} B_{i,r} < SBR$, that is when the neighbors of r do not manage to provide a sufficient amount of stream data to r [69]. The reasons for this starvation include, as we said above:

- Content Bottlenecks: the sender neighbors are not able to send enough chunks to the receiver *because they do not have any chunks in their buffers that could be useful to* r.
- Bandwidth Bottlenecks: all senders combined can offer a *total bandwidth which is too low for the receiver to be able to retrieve the stream at a rate of SBR*, even if they hold chunks which can be useful to r.

The content bottleneck problem can be limited by choosing an appropriate chunk selection algorithm, which should make sure that data is uniformly spread among nodes, so that they can exploit at best their upload capacity. The bandwidth bottleneck problem can be addressed (within the framework of the Edmonds' theorem) by using an appropriate peer selection algorithm and adequate values for the node out-degree.

From an external point of view, the effect of both forms of bottlenecks is the same: the stream is delivered to r at a lower rate than expected. But the distinction between content bottlenecks and bandwidth bottlenecks is a useful one to be made when studying and optimizing the performances of a data-driven system, as it may allow to isolate the shortcomings of a given chunk-selection algorithm from those due to a poor peer selection strategy.

The Resource Index Describing the availability of bandwidth resources on a macroscopic scale can be useful to quickly evaluate the overall serving capacity of a system and to identify excess or shortage in bandwidth resources. A simple way to do so is to calculate the *Resource Index* (RI) of a given streaming session [104]. The RI is defined as the ratio between the available serving capacity of the system and the capacity required to fully serve the node population. In formula, when there are N peers (excluding the source) with bandwidth U and if the stream rate is SBR:

$$RI = \frac{U_{SRC} + \sum_{i=0}^{N} U_i}{N \cdot SBR}$$

We notice that, by its definition, $RI \geq \xi$ for any distribution of node bandwidth. A resource index larger than one indicates a global bandwidth excess, whereas values smaller than one indicate that there are not enough resources to serve all the peers in the system. The most challenging bandwidth scenarios are those where the RI is only slightly higher than one, but every node is expected to receive the full streaming service.

5.2.3 Understanding the Data Distribution Process

In data-driven systems, as stated above, data exchanges are performed among the nodes based on local criteria, such as the current state of data distribution (reconciliation of the buffer content at neighboring nodes), the existence of established neighboring relationships with other peers (data connections and control information), and the local knowledge about the past evolution of the system. For this reason, even a detailed knowledge of the mesh structure - which gives only an approximate short-term characterization of the system activity - is not sufficient to capture the the details of the data distribution process.

A more precise understanding of a data-driven system can be gained from the stochastic analysis of the paths taken by data chunks. We recall that these paths are spanning trees, that is acyclic[6] directed graphs connecting all nodes that received a specific chunk. It is possible to look at the data distribution performance in a different way by studying and measuring the average performance of chunk distribution trees, such as the tree depths and the average layer out-degree: this kind of analysis is particularly useful when the environment in which the system runs is not fully known a priori (for instance, if nodes have an unknown bandwidth distribution).

Average Maximal Depth of Distribution Trees Maximal tree depth $\mathcal{D}(c_j)$ is the number of hops required for a chunk c_j to reach from the source S the totality of the N' nodes that actually received it ($N' \subseteq N$). In formula:

$$\mathcal{D}(c_j) = \max_{i \in N'} \; hops(\, S \; \xrightarrow{c_j} \; i \,)$$

We must briefly emphasize here the difference between lag and tree depth: while the former also takes into account the time required for chunk delivery, tree depth only deals with the maximum number of hops that a single chunk travels. We remember that each node autonomously decides the next data chunk to request using the *chunk request* algorithms based on locally available knowledge. Since a chunk can typically only be sent following a request, the actual delay introduced by a hop also depends on the timing with which the request was answered by a serving node.

While small tree depths do not directly imply that data distribution is efficient (e.g. in the case of widespread chunk loss), this metric can however give an idea of the *maximum number of exchanges* needed to distribute a chunk. To increase the meaningfulness of the observed value, it is advisable to compute the average tree depths on several (k) contiguous chunks. We define the Average Tree Depth metric for chunks c_n to c_{n+k} as:

$$\mathcal{D}_{avg}(c_n,\, c_{n+k}) = \frac{1}{k+1} \sum_{i=n}^{n+k} \mathcal{D}(c_i)$$

[6] We assume for simplicity that chunks are not unnecessarily duplicated, which means that there are no loops in the distribution graph.

5.2.4 Locality Awareness of Data Exchanges

The fundamental strength of native network-layer multicast lies in the fact that the path taken by the data is always the shortest: the routers taking part in a multicast session are only those that lie on the direct path between the source and all the destinations. This results in two desirable properties: the optimal latency of the data paths and the optimal use of the available capacity of the network.

Application-layer multicast cannot match the optimal performance of network layer multicast. In the previous sections, we have discussed at length about the allocation of upload bandwidth capacity at the access link. However, we must not forget that data travel on the transport network over multiple unicast connections between the nodes. This means that *the network path between the source and the destination is not the shortest one*. Nodes establish connections without any knowledge of the underlying network topology, so it is not uncommon for data to travel multiple times over backbones even if the shortest path between the source and the destination would not cross the boundaries of a same AS.

The earliest application-layer multicast designs were very much concerned with this inefficient use of the network [30][12][110]. The performance of these overlays was evaluated using metrics such as the *network stretch*, defined as the multiplicative ratio between the length of an overlay path and of the unicast path connecting a node to the source (also known as RDP - *relative delay penalty*), and *link stress*, defined as the number of copies of the same data that are sent over the same physical link. These metrics need the full knowledge of the network topology to be computed, and are very useful to determine the scalability of a given static overlay architecture by means of simulation.

However, these metrics are not applicable in a straightforward way to data-driven systems, as the overlay structure is not fixed. A possible solution would require analyzing a number of subsequent chunk distribution trees, from which the stress and stretch metrics could be computed, and then averaging those figures to come up with the typical behavior of the overlay construction algorithms. Problems arise though when the data-driven system is dynamic, as its evolution depends both on the current status of the network and on external factors, such as resource availability and distribution of node pairwise latencies.

We argue that the main problems data-driven systems intend to solve are the robustness against node churn and the allocation of the available upload capacity, rather than the achievement of optimal delay performance: it does not make a lot of sense to compare a data-driven system to a fixed-structure overlay to determine which system is better, as fixed-structure overlays surely offer better performance in terms of delay.

Average Latency of Node Connections The main information available to an application about network locality is the *end-to-end latency* of a transport connection. Several techniques exist to measure latency: the simplest is based on the *ping* principle, that is measuring the round-trip-time (RTT) required for a short message to reach another node and to be retransmitted back. To improve the accuracy of the measurement, more complex techniques are used which aim to factor out the delay introduced by processing time at the remote node (for example, the NTP host synchronization protocol uses this approach). A common approximation to estimate the one-way connection delay is to just divide the RTT by two, even if the forward and backward paths could well be asymmetric.

Nodes in data-driven systems associate to exchange both data and control information. The choice of partner nodes is important for two reasons:

- control messages can be sensitive to delay, as they may contain information that tends to evolve rapidly over time. A control message can lose accuracy if its delivery is delayed for too long, and the actions performed in response to an outdated message can result in a waste of bandwidth (transmission of duplicate chunks, etc.)
- data transfer over high-latency connections means an inefficient use of the underlying network, as it results in both high stretch and stress.

For instance, it can be interesting to evaluate the locality awareness of a data-driven system by performing an analysis of the average latency of the connections that a peer chooses to establish when the system has reached steady state.

Amount of Data Transferred vs. Transfer Locality As the quantitative weight of data transfers in a streaming application is much higher than the weight of control traffic, improving the locality awareness of data transfers is a key requirement to avoid an inefficient use of network resources. We find it especially interesting to analyze, from the point of view of a single node, how many data were exchanged with "nearby" peers, compared to how many data were exchanged with nodes located far away: this information is not adequately conveyed just by the average latency of the *connections established* by a node, as there is no notion about the amount of data transferred.

The technique we use to define the average locality of data transfers is the following: at each node, we calculate the number of chunks sent to each of its peers over a given time interval. Then, based on the pairwise latency matrix (which can obtained from measurements performed while the system runs), each node defines a number of *latency bins* of uniform width. Each node computes the value of the amount of data associated with each latency bin by computing the total amount of chunks sent to nodes whose link latency falls in the range of each bin. Then, we aggregate the values contained in the same bin for all the nodes in the system, dividing by the number of the nodes. Finally, the value of the bins is again normalized by the total amount of data chunks generated by the source over the time interval. The result of this process can be represented as an histogram correlating the percentage amount of data exchanged in average to the *distance* it has traveled on the network (Figure 5.4).

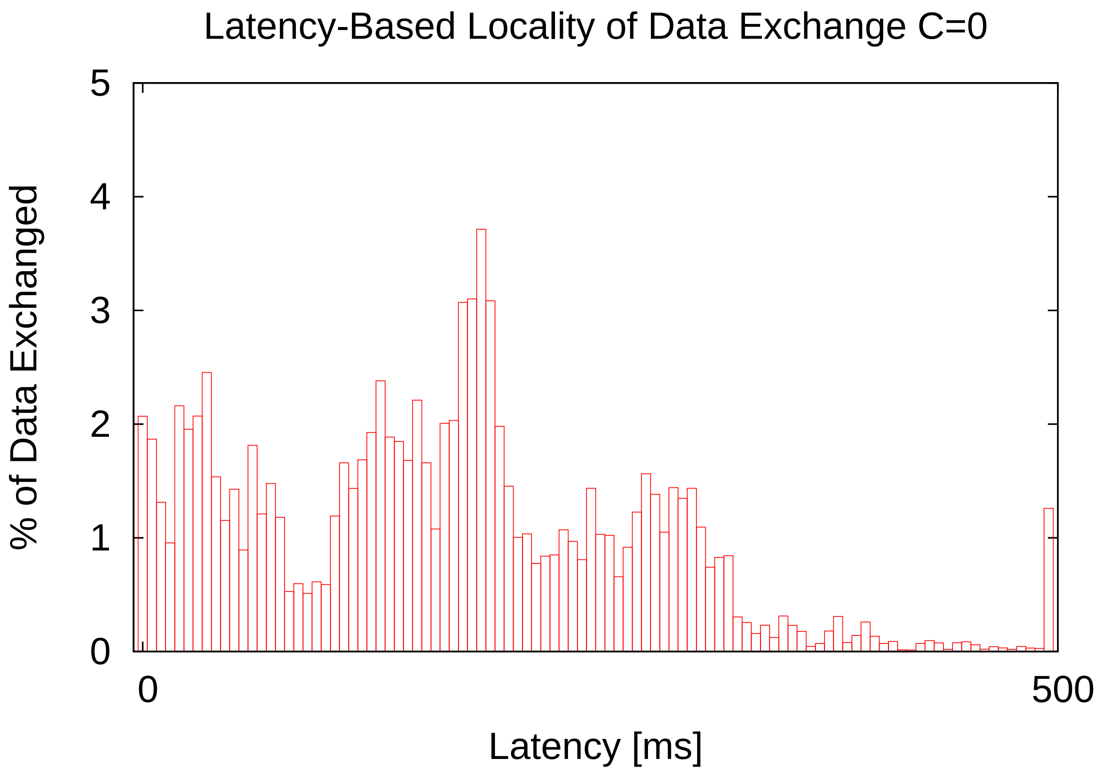


Figure 5.4: Amount of Data Transmitted versus Transfer Locality

Locality and Average Node Lag Finally, we can easily expect that the latency of the individual data transfers will have a cumulative impact on the overall delay with which chunks propagate through the system. However, as we said before, this overall delay cannot be compared to the delay of the unicast path from the source to the node (i.e. to compute network stretch): the average chunk reception delay, or node lag, is not only due to the performance of individual links, but its main component is made up by the scheduling decisions and node associations independently performed by all the peers.

While a comparison of node lag to the “absolute” reference of unicast delay is meaningless, the fact that node lag is influenced by the awareness to network locality allows us to establish “relative” comparisons between systems. By looking at the average lag, we can draft a rough comparison of the locality awareness of different data-driven systems that operate in similar conditions. However, this metric is useful to evaluate the effect of slight parametric changes in the algorithms of a same system, giving the feedback required to experimentally tune the algorithms to better operate under real-world conditions.

5.3 Behavioral Metrics: the Role of Incentives

We now focus our analysis on PULSE-specific mechanisms and properties. PULSE is basically a data-driven system, but its algorithms present additional features that this general definition does not capture:

- First, the choice of partners for data exchanges is determined by the concept of *lag*, a metric that describes the stream reception status of each node. As the data retrieval process is restricted to a small continuous segment of the stream (the *trading window*), nodes use information about *lag* to discriminate between the nodes that may reciprocate and those that cannot.
- Second, being an incentive-based system, the recent history of the data distribution process contributes to the choice of possible targets for node associations. Also, the optional altruistic mechanism exploits long-term knowledge about remote node behavior as a bias for peer selection.

In this section we introduce three new metrics to better understand the behavior of PULSE, analyzing it under a different point of view. While the metrics we described above capture the external aspects of the system - such as the shape of its mesh, its observed loss rate, or the average reception delay - we now start correlating the system performance and the decisions taken at the individual nodes to their own and their partners' resources, i.e. mainly to their upload bandwidth.

The first metric, *affinity*, deals with the effectiveness of the peer selection algorithm for the MISSING connections. The second, *friendliness*, tries to portray the role of altruism inside the system. The last one, *soft fairness*, aims to capture the reaction of the system to the bandwidth conditions in which it happens to run.

5.3.1 Class Affinity

A useful way to capture the behavior of a given bandwidth class can be the frequency at which its nodes establish MISSING links toward nodes of the same class, or to any other class.

The choice of neighbor nodes for MISSING data exchanges is in fact critical to the good behavior of the system, as MISSING partnerships should convey the largest share of stream data to most resourceful nodes. It is thus very important for these partnerships to be established *with resourceful nodes that share a common data interest range*, to maximize the reciprocal benefit both peers can obtain from the relationship.

We thus define the concept of "Class Affinity" between class α and class β as:

$$\Phi(\alpha,\beta)(t) = \frac{\sum_{n\in\alpha} \|n\text{'s MISSING links toward nodes of class } \beta\|}{\sum_{n\in\alpha} \|n\text{'s total MISSING links}\|} \tag{5.1}$$

We will also represent in our plots a normalized version of the Class Affinity metric. The purpose of normalization is to take into account the different cardinality both for the different bandwidth classes and for the different scenarios. To this end, we divide the value of Φ by the fraction of the total population belonging to the target class. In other words, this amounts to comparing the choices performed by a node with the *uniformly random* node selection policy.

$$\Phi'(\alpha, \beta)(t) = \Phi(\alpha, \beta)(t) \cdot \frac{N}{\|\beta\|} \tag{5.2}$$

An interesting way to represent the internal cohesion among nodes from the same class can be captured by the "self affinity" $\Phi(\alpha, \alpha)(t)$ of any given class. Intuitively, this coefficient will be higher when most nodes from one class have many connections to nodes of the same class. This would be an indicator of a heavy reliance of any group of nodes on nodes with a similar bandwidth capacity. Affinities toward other classes, richer or poorer, are also interesting to analyze under a different light how nodes tends to negotiate their main way of obtaining chunks.

Interpretation We expect rather high "self affinity" results for classes with considerable excess resources in highly heterogeneous simulation scenarios. On the other hand, results showing that the affinities between different classes are very similar could indicate that tit-for-tat is not the dominant criterion impacting MISSING peer selection: typically, this should be the case in scenarios with little resource heterogeneity between bandwidth classes. Also, by comparing these plots with the *node lag* graphs we can correlate variations in the affinity metrics to changing system conditions (e.g. convergence, steady state, etc.).

5.3.2 Class Friendliness

Another interesting piece of information is the behavior of FORWARD connections. We recall that the purpose of FORWARD connections is to better utilize the capacity of richer nodes to help other nodes in the system, while avoiding to negatively impact their MISSING exchange performance.

We thus define the concept of "Class Friendliness" between class α and class β as:

$$\Psi(\alpha, \beta)(t) = \frac{\sum_{n \in \alpha} \|n\text{'s active FORWARD links toward nodes of class } \beta\|}{\sum_{n \in \alpha} \|n\text{'s total active FORWARD links}\|} \tag{5.3}$$

As we did above, we will again normalize the metric against the fraction of the nodes belonging to the target bandwidth class:

$$\Psi'(\alpha, \beta)(t) = \Psi(\alpha, \beta)(t) \cdot \frac{N}{\|\beta\|} \tag{5.4}$$

Class Friendliness is similar to Class Affinity in that it describes the likeliness of interaction among node classes, but since FORWARD connections are not really active until a node actually sends data over them, Ψ has to be computed at the end of each peer's EPOCH.

Interpretation We expect that normalized friendliness at steady state will be higher toward classes with decreasing bandwidth resources. We are thus interested in analyzing the evolution over time of $\Psi(t)$, especially during the system convergence phase and following major changes in the network configuration (spike of arrivals, etc.).

5.3.3 Soft Fairness

The word *fairness* is often employed to describe a vague desirable property of a system that is somewhat along the lines of 'who gives shall receive'.

This definition may be enough for conveying a general idea of what fairness is about. However, when comparisons have to be made between the fairness of two systems (or of the same system for different initial conditions), this definition is clearly insufficient. How fair is a system? How can fairness be measured? And then, what to do when a node does not 'give'? If a system - in presence of a sufficient service capacity - does exclude the nodes who give less than required, is it 'more fair' than a system that serves them anyways?

About the Meaning of Fairness We feel that giving a clear definition of what fairness means is a problem in itself, one that has deep roots outside the scope of computer science. We do not wish to deal here with the underlying philosophical questions of "what's the right/wrong behavior and how right/wrong deeds are rewarded" applied to a rational agent's behavior inside a generic cooperative system. We will however observe that the definition of fairness used by each author is deeply influenced by his "moral" evaluation of the fairness problem, within the specific constraints of the application at stake.

Some authors thus understand *fairness* as *"equal contribution by each node in the system"*, which has to be either expected [12][110], required [21], or enforced reactively [89][107][46]. This concept is sometimes expressed in a different (but closely related[7]) form, "*the service received from the system is proportional to the node's contribution*", e.g. in BitTorrent [33].

Contributions may consist in actions performed by a node on behalf of another one (e.g. forwarding a message to a third party), services offered directly to the requesting node (e.g. upload of a requested file), and/or offers of future service (e.g. replying to a query). Finally, enforcement can be performed on the short term (bit-torrent's tit-for-tat, bit-for-bit) and/or on the long term (cumulative tit-for-tat, reputation, trust).

[7]Game theory allows to reduce a game with more than two players to the same game between one player and "the rest of the system".

This understanding of fairness is well-suited to applications with an implicit incentive for *long-term abuse of the system*. For example, in a file-sharing system, it may be reasonable to lower the priority of requests coming from nodes that share fewer files, to prevent them from clogging up the service queues at the other nodes – thus protecting the performances of those peers that share more. Another example can be a large-scale content distribution network: it may be reasonable for nodes to serve data to peers that do likewise contribute service capacity than to peers that never give back.

The applications that are well-suited by this type of fairness often share one or more of the following aspects:

- The service performed has a finite duration (e.g. transfer of a file), after which the requesting node is no longer interested in staying in the system.
- The *gain* a node can achieve from the network, is inversely proportional to the time it spends in the system.
- The *gain* of a node is not upper-bounded as a consequence of the type of resources that are served (e.g. a huge number of different files vs. a single file).
- The value of the objects served by the system does not decrease over time, so that it becomes possible to "profit" by accumulating them.

Defining a Metric for Fairness To measure fairness following the above definition, we need a metric that can take into account both the contribution of a node (i.e. its available outbound bandwidth) and the local outcome it experiences. In PULSE, we can consider the local outcome as the inbound bandwidth a node manages to obtain from its peers. If we take into account the fact that the streaming application is rate-limited, we can then describe the amount - and the steadiness - of a node's incoming bandwidth by the *position* (i.e. the average node lag) that it occupies inside the system.

Our definition of "Soft Fairness" (bandwidth class version) between any two *bandwidth classes* α and β, whose upload capacity is $U_\alpha < U_\beta$, is :

$$\mathcal{F}(\alpha,\beta)(t) = \frac{\sum_{n\in\alpha}\sum_{m\in\beta} \mathbb{I}(\, T_B(n)(t) \leq T_B(m)(t)\,)}{\|\alpha\|\,\|\beta\|} \tag{5.5}$$

where the indicator function $\mathbb{I}(x)$ is defined as

$$\mathbb{I}(x) = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

and T_B is, as usual, the average node lag measured at time t.

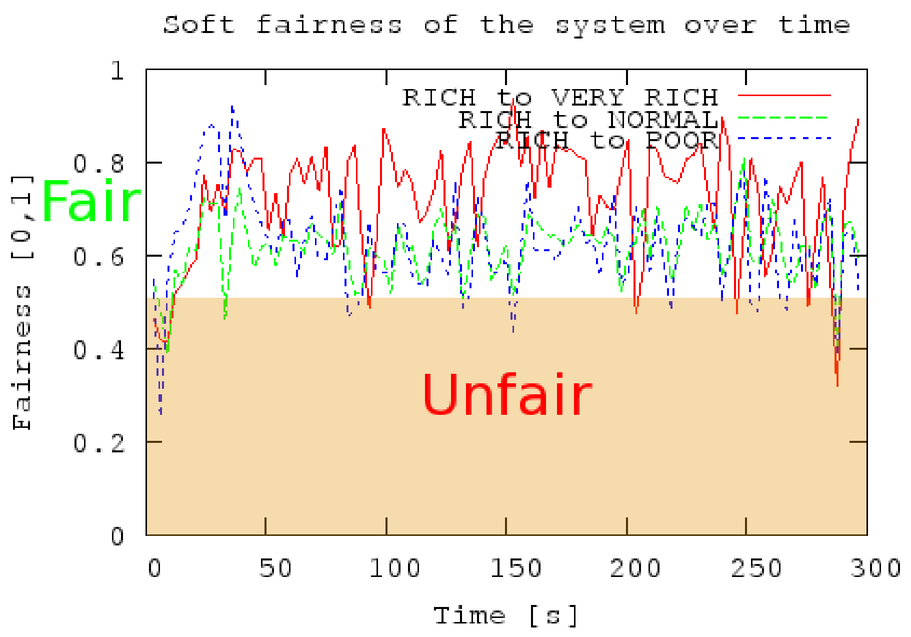


Figure 5.5: Soft Fairness Plot for a Given Bandwidth Class (in this example: RICH)

Should bandwidth classes not be defined (for instance, in a real system where the possible U for each node can take a wide range of values) the same "Soft Fairness" for this general case would become:

$$\mathcal{F}(t) = \frac{\sum_{n \in N} \sum_{m \neq n : UB(n) \leq UB(m)} \mathbb{I}(\, T_B(n)(t) \leq T_B(m)(t)\,)}{N(N-1)/2} \tag{5.6}$$

Values of $\mathcal{F}$ near 1 mean that the system is "fair", since the near totality of nodes that contribute more to the system get a steadier incoming bandwidth than less-contributing ones, allowing them to settle on a lower lag value than poorer classes. On the other hand, values near zero indicate that the system is "unfair", since those who contribute less can systematically get in return the needed data chunks with a better lag performance. Finally, intermediate values could suggest that there is no strong correlation between the capacity provided by a node class and the lag it manages to obtain.

Graphic Representation We will represent the class-based fairness values on a plot like the one shown in Figure 5.5. In this kind of plot, to include the 'reverse fairness' relationships between classes with higher-to-lower upload, i.e. $U_\alpha > U_\beta$, we will represent the function:

$$\bar{\mathcal{F}}(\alpha, \beta)(t) = \frac{\sum_{n \in \alpha} \sum_{m \in \beta} \mathbb{I}(\, T_B(n)(t) \geq T_B(m)(t)\,)}{\|\alpha\|\, \|\beta\|} \tag{5.7}$$

Applying this convention, we can obtain pictures where it's possible to grasp immediately the amount of soft fairness relative to a pair of bandwidth classes. The plot can be then read as if divided into two horizontal stripes that correspond to two zones of "prevalent fairness" and "prevalent unfairness", as we explained above.

Interpretation Soft Fairness is an useful metric to describe a running PULSE system. It allows to compare the average performances observed by nodes with different resources. This can offer an empirical description of the effectiveness of the tit-for-tat retribution mechanism in a given operating scenario. Intuitively, we expect tit-for-tat to be more effective when operating in highly heterogeneous conditions and with scarce bandwidth availability, and less relevant whenever bandwidth is abundant.

5.3.4 Toward a Better Concept of Fairness for PULSE

We introduced the *soft fairness* metric to assess the relationship between a peer's bandwidth contribution to the system and its lag. This metric is especially relevant when the resources of the system are scarce (RI close to 1 or even lower). On the other hand, when there is no scarcity, the retribution mechanisms will be less effective in discerning the nodes that give more from those that give less. This is also what we intuitively expect from a rate-limited application

However, we must not forget that fairness as "just retribution" is not our primary interest. After all, a live streaming application is primarily meant to distribute streaming data. Incentive mechanisms are important to make the system resilient to peers who are uncooperative - either for technical limitations or deliberate choices - since spontaneous cooperation should never be expected. However, their role and usefulness of an incentive mechanism is subordinated to the system's main purpose, that is *delivering a live stream to large audiences with low delay*.

In the context of our target application, we feel that fairness described as in Section 5.3.3 - either same contribution for all nodes, or service provided to each node by the system is equal to the node's contribution - is still not totally appropriate to evaluate a live streaming application, for the following reasons:

- Live streaming is a rate-limited application, meaning that there is little interest for each node in obtaining at steady state *more data* (i.e. at a faster rate) than other nodes.
- No node can in any case receive data faster than the source produces them. The interest function (i.e. reception delay) is now upper-bounded.
- Live streaming deals with "ephemeral" content, whose value tends to zero over time.

Moreover, some constraints of live streaming and some properties of the current Internet also show the limits of that fairness concept:

- Studies about the availability of bandwidth resources at Internet hosts show how their distribution is not uniform throughout the entire population, but more similar to a truncated power law [96][104].
- Systematically enforcing strict limits on the instantaneous bandwidth (e.g., because of long-term considerations) can hurt the short-term performances of the system, especially when the usefulness of the content is so short-lived. As we know, bandwidth may be "infinite" over time, but it is limited at any given moment: we can thus think at unused available bandwidth as if it was *lost service capacity* [107].
- And finally, using up a larger share of the bandwidth available at a node does not imply (in most cases) an additional cost.

We will therefore suggest our own concept of fairness, which is more oriented to the common interest and survivability of the whole system rather than the interest of the individual nodes. We attempt here a definition of "*global fairness*" as *the property of a system where 'good' actions of its individual components give origin to 'good' local outcomes and 'good' global outcomes, and where 'bad' actions of its individual components may give origin to 'bad' local outcomes, but also lead to 'good' global outcomes*.

Applying this definition to PULSE, we can say that a good (bad) action is (not) offering enough upload bandwidth, i.e. less than the stream bit-rate SBR. A good (bad) local outcome at each node is receiving (less than) an average stream bit-rate of SBR. The good global outcome is that *as many nodes as possible are served by the system* – regardless of the serving capacity distribution across the population. This new definition of fairness better translates, in the specific context of our application, to the need for the system to "protect itself against exploitation", specifically by nodes that cannot / don't want to contribute enough. As long as there is enough serving capacity, we do not feel that "poor" nodes or freeloaders should be penalized. On the other hand, when the service capacity becomes scarce, the first nodes to be penalized should be those that are contributing the least, in name of the "common good" of the system [83].

5.4 Conclusion

In this chapter, we provided two sets of metrics to evaluate mesh-based live streaming systems. The first set, based on the concept of *lag*, is intended as a common framework that may be used to describe and compare the performance of generic data-driven systems. The second set is designed to study the macroscopic relationships between node resources, their placement in the system, and their data reception performance: while possibly useful in a more generic context, these metrics specifically aim to capture the internal dynamics of the PULSE system and to monitor the evolution of node placement inside the network.

The metrics we presented in this chapter are in no way meant to be exhaustive. We believe that other new metrics could be introduced to study certain specific aspects of data-driven systems:

for instance, further insights could be gained from the measurement of graph properties of the mesh overlay (e.g. clustering coefficient, diameter, etc.). We decided, in the context of our study of PULSE, not to focus too much on the full overlay graph, as it keeps changing rapidly over time, but rather to privilege the study of data exchange paths supported by the statistical analysis of node relationships.

Chapter 6

Simulation Results

In this chapter[1], we evaluate the performance of PULSE algorithms through extensive simulations. We introduce in Section 6.1 the simulated model in full detail, underlining its strengths and shortcomings. In Section 6.2 we describe the simulation scenarios used, while in Section 6.3 we analyze the stability and efficiency of the simulated system over a range of structural parameters, such as length of the Trading Window and amount of FEC added by the source. In Section 6.4 we apply the metrics we presented in Chapter 5 to understand the global behavior of the peers and to describe the internal dynamics of the PULSE system. Section 6.5 examines the behavior of PULSE under the various scenarios we introduce in this chapter, with a specific interest for the quantitative performance of the system. Section 6.6 examines the effects of churn and sudden variations in the membership and in the available resources while the system is operating. Finally, we conclude our analysis with additional comments in Section 6.7.

6.1 Methodology and Expectations

PulSIM, the simulator we implemented and used to obtain these results (Chapter 4), is a simple time-slotted simulator. The first design choice which is fundamental in order to perform a realistic time-driven modeling of any phenomenon is the time unit (or *step duration*) of the simulation: while a *too small time unit makes the simulation slow*, increases the volume of the data produced, with little benefit on the accuracy of the data itself, choosing a *too large timescale can give inaccurate results*. Then, the accuracy of the model has to be perfected depending on the selected time scale: this includes a *model of data transfer at the chosen time scale* and the *definition of a knowledge propagation model*: these rules basically regulate what pieces of information are available to which node with how much propagation delay. Finally, based on the modeling choices, it is advisable to formulate an *expectation about the overall accuracy of the model*: this should give a clear idea of what will be the limited scope inside which results will be valid, and some predictive reasons on why and when the model could deviate from reality.

[1]The contents of this chapter have been published in part as [87].

6.1.1 Choice of Simulation Step

Simulating a system at a certain timescale implies the ability to sample its state with a period that cannot be smaller than the simulation step. By Nyquist's Theorem, the lossless sampling of a system requires a frequency which is at least twice the signal's frequency. However, we must not forget that the simulation step also determines the maximum frequency at which the simulated system can act: therefore, the simulator frequency should be a multiple of the double of the actual rate at which things evolve in the system. An appropriate timescale can usually be chosen by selecting *a step value about one order of magnitude smaller* than the actual time constant of the phenomenon.

The best way to choose a correct time unit is to examine the system that has to be modeled and recognize the frequency at which important events happen. In PULSE, the peer selection is performed at regular intervals of several seconds, called EPOCHs[2]. Peer selection is based on the actual performance of data exchange during the previous EPOCH, so the simulator has to allow for repeated exchanges to take place between two subsequent peer selections. The rate of data exchanges will be on average in the same order of magnitude of the data generation process, as live streaming is a rate-limited application. Therefore, we believe that a time step in the order of the tenths of seconds would be a good choice. In our simulations, we used a value of 0.25 sec.

6.1.2 Modeling Data Transfers

We now concentrate on the way to model data exchanges in a data-driven system at the chosen time scale. We remember that the basic unit of data exchange in our context is the *chunk*: new chunks are generated by the source at a constant rate *CR*. We suppose that the size of a data chunk can range between few KB to few hundreds KB, depending on the actual rate of the stream and on the amount of error correction protecting the stream data. The issues then becomes: how can we allocate node capacity to simulate data transfers in a realistic but lightweight fashion?

First, we can introduce one of the traditional hypotheses, which has been used thoroughly in the modeling literature: that the bottleneck link is always located at the edge of the network, i.e. each node is bandwidth-constrained but the network does not otherwise affect the data exchange. This results in a simple transit-stub network topology, where the transit has an infinite bandwidth and the stubs are dimensioned to approximate realistic access link capacities. We also suppose that download capacities are always higher (e.g. twice at least) than the stream rate.

Then, we have to face the problem of how to model the behavior of our network: we are especially concerned about the transfer delays between nodes and about pairwise node latency. The network should then introduce an initial delay for any communication to be possible between two nodes, on top of which it adds the delay required to transmit the data payload. But how can we model data transfers, which in reality are a continuous phenomenon, in the context of a time-driven simulator?

[2]The typical EPOCH length that we used in our simulations is two seconds.

To simplify things, we can think about having data transfers partially synchronized across the system, so that they always respect the boundaries of a time step. This can be done (for instance) by limiting the total number of chunks that a node can generate during each step. The amount of upload/download bandwidth at each node determines the amount of chunks that can be exchanged by a node[3] during each time step. Having introduced this further approximation, we can be sure that all the chunk transfers will be completed on time (if the full capacity is used) or before the end of a step (if spare capacity remains), but never later.

The simplification above leads us toward the concept of *bandwidth slot*, which constitutes the practical unit of measure to represent data transfers in our model. A bandwidth slot is defined as the amount of bandwidth required to transmit one chunk in a single time slot. Node capacities are then defined in terms of multiple bandwidth slots: this quantization of bandwidth capacities can approximate quite well the average behavior of TCP under congestion at the edge, since the available upload node capacity during a time step is equally shared by all the competing chunk transfers.

When dealing with latencies, the relatively large time scale we selected for our step value comes into play with a positive outcome. If we consider the typical pairwise latency values measured over the Internet, which range from few tens to few hundreds milliseconds, we can approximate the maximum time a message requires to reach another node with the duration of a whole time step. This is especially useful for control messages, which are quite short (tens of bytes) and whose transfer delay is mainly dominated by the pairwise latency between nodes. The fact of expressing pairwise latencies in terms of single simulator steps helps us to model the evolution of the internal state of the system, as state changes induced by control messages become effective with an uniform delay of one iteration.

6.1.3 Model of Knowledge Propagation

Under the above assumptions for data and control exchanges, we can build a knowledge model which is suitable to simulate the internal state of PULSE nodes. In the real application, nodes obtain knowledge about the rest of the system by randomized gossiping and by direct exchanges of control messages. On the other hand, we do not wish to implement a simulated gossiping process, as this would increase by far the complexity of our simulator (the interactions of the gossip protocol with the actual PULSE control exchanges would be difficult to understand and debug).

For this reason, we approximate the propagation of information in the system in the following way:

1. BLUE knowledge: at each iteration, *all nodes* are aware of the $T_{B_{avg}}$ at the previous iteration *for all other nodes*. This oracle-like source of knowledge replaces in the simulation model the actual gossip protocol. We argue that, when system reaches steady state, as the

[3]This limitation holds both for uploads and downloads.

values of $T_{B_{avg}}$ are substantially stable, the fact of having an updated knowledge of average remote node positions approximates what happens in reality, i.e. low-frequency updates of a $T_{B_{avg}}$ value that is slowly changing over time.

2. RED knowledge: it is critical for the realistic modeling of the system that the detailed knowledge of each peer's buffer be available only to a small, well-chosen subset of the entire node population. In this case, we need to roughly simulate the inner workings of the PULSE protocol and cannot rely on an oracle-like mechanism as we did above. For this reason, we have to specify few simple rules that constrain RED knowledge propagation.
   A node P is aware of the full buffer state of another node Q at the previous time step, including any chunk requests it may have expressed, if and only if:

   (a) Q appears in the MISSING / FORWARD neighbor list of P

   (b) P appears in the MISSING / FORWARD neighbor list of Q

   (c) Q has chosen to send a RED control message to P at the previous time step

   (d) P has chosen to send a RED control message to Q at the previous time step

3. DATA knowledge: nodes become aware of the chunks they received from their neighbors at the iteration following the data transfer. Moreover, a mechanism is in place to prevent the synchronous transfer of the same chunk by several different nodes at the same iteration: in these cases, either the next queued chunk is sent (instead of the potential duplicate), or the node is skipped altogether, when there are no more chunks in the request queue.

6.1.4 Expectations and Limits of Our Modeling Approach

The choices we made while formulating our model of the PULSE system are all aimed to provide a faithful description in terms of internal dynamics and global behavior at steady state. However, the compromise we had to reach between simplicity and faithfulness has several consequences on the kind of results we can expect from PulSIM: we try to synthesize our choices in the pages that follow.

- Since all nodes are modeled as having an independent internal state, and since each of them has to apply the PULSE algorithms on its data structures, the computational complexity of the simulator is rather high. While we haven't performed a thorough study of the exact dependence between every system parameter and the time required for a complete simulation run, we can say that the overall complexity roughly scales:

  1. linearly with the duration of a simulation run
  2. quadratically with the size of the buffer window TW and the chunk rate R
  3. linearly with the total number of nodes in the system

While the memory footprint of the simulator can be easily contained in the RAM of a recent computer (e.g. about 1GB of memory used by our largest simulations with 10^4 nodes), the actual bottleneck was given by the available CPU power. We had to resort to run several instances of the simulator in parallel to minimize waiting times: on the machine we managed to obtain to run simulations, a dual dual-core Intel Xeon server clocked at 2.8GHz and with 7GB RAM, a "simulation unit" made of 16 different scenarios (serially run over four independent threads) would take from less than two days with 10^3 nodes, to more than one week with 10^4.

- The bandwidth granularity of the simulation is implicitly determined by several parameters. There is a relationship binding together stream rate (SBR), chunk size (CS), chunk rate (R), time step duration ($STEP$), and maximum bandwidth slot size (BS_{max}):

$$BS_{max} = \frac{SBR}{CS} \cdot STEP = R \cdot STEP$$

  The smallest bandwidth granularity should be dimensioned to be a fraction of (or at least equal to) the smallest possible peer upload bandwidth. An important consequence of our design choices is that the simulation results depend only on chunk rate, simulation step, and bandwidth slot size. It is thus possible to rescale the results of a single simulation to fit different values of these parameters through this relationship. Typically, we can scale up the stream rate by scaling up the chunk size, double the chunk rate by halving the time duration of a simulation step, etc., but we must be careful to always stay within the reasonable range of operation given by the other simulator assumptions detailed above.

The model works very well to study the behavior of the system at steady state. It is conceived to evaluate with particular accuracy the impact of different bandwidth scenarios on the evolution of the system as a whole. However, there are also some drawbacks:

- The simulation model does not include a way to define latency scenarios other than the uniform latency distribution. This does not allow us to evaluate through simulations the behavior of the PULSE algorithms with respect to pairwise node delay and network locality in general.

- The simulation model is also not suitable to predict the amount of bandwidth overhead brought by control traffic, nor its possible interactions with the actual streaming traffic and its impact on the operation of rest of the network.

- Another weakness of the model is in its simplified knowledge model, which overestimates the likelihood of chunk exchanges between nodes and the efficiency in the use of available upload capacity when compared to reality.

- Since the evolution of the simulation is not tunable by the operator once a run has been launched, and since the simulator relies on randomized oracle-like mechanisms to model less-relevant aspects of the system, simulations may sometimes produce artifacts. Especially when modeling concurrent node arrivals, it is not uncommon to encounter a certain degree of artificial transitory instability.

- The node reception lag results, as output by the simulator, should not be expected to be faithful “in absolute” to measured results from real-world systems, as they do not take into account with sufficient accuracy the delays induced by the exchanges of control information which always happen before the transmission of the stream data. Also, as the real throughput of data exchanges depends on TCP and on the underlying network topology, the worst-case duration of a chunk exchange is not limited to the time step duration. Finally, the real system is asynchronous, so its actual performance will surely be inferior to its synchronous approximation attempted by this model.

The primary goal of our simulation model is to preserve the *system-wide order* of the nodes in terms of reception lag. This goal is motivated by our main hypothesis, namely that the lag of the members of each bandwidth class is in some way dependent on the total (system-wide) and relative (class vs. class) availability of resources at the nodes. The PulSIM simulator will be our main instrument to validate whether this conjecture holds both at steady state and during system convergence.

6.2 A Set of Scenarios for Simulation

We now set out to evaluate the behavior of nodes that concurrently run the PULSE algorithms. As the algorithms are designed to adapt to the bandwidth conditions in the system, choosing appropriate bandwidth scenarios that represent a wide range of possible operating conditions will be crucial to understand how they work. Also, we want to have get some clues on how the system is able to react to churn: for this reason, we apply several synthetic arrival and departure patterns to the node population, to test the system’s response to slow and sudden membership variations.

Bandwidth Scenarios The simulator parameters and bandwidth distribution ranges have been chosen to model the diffusion of a 1 Mbps FEC-protected stream. In all the scenarios we use, the source’s maximum upload bandwidth is set to 3*SBR. The values of the Resource index (RI) for each scenario do not include the source’s bandwidth since it would make the RI dependent on the population size. However, if we consider a population of 1000 nodes, the RI increase introduced by the source is just 0.003. The various scenarios are also summarized in Table 6.1.

High Heterogeneity, Low Bandwidth (HH-LB) This scenario encompasses four bandwidth classes: 4% of VERY RICH peers, with 4*SBR upload and 4*SBR download bandwidth; 20% of RICH peers, with 2*SBR upload and 2*SBR download bandwidth; 21% of NORMAL peers,

Class Name	HH-LB	HH-HB	LH-LB	LH-HB
VERY RICH (VR)	4%, 4*SBR	4%, 10*SBR	=	=
RICH (R)	20%, 2*SBR	20%, 3*SBR	20%, 2*SBR	20%, 4*SBR
NORMAL (N)	21%, SBR	21%, SBR	80%, SBR	80%, SBR
POOR (P)	55%, SBR/2	55%, SBR/2	=	=
Resource Index (RI)	1.045	1.485	1.2	1.6

Class Name	UNIFORM-LB	UNIFORM-HB
NORMAL (N)	100%, SBR	100%, 1.5*SBR
Resource Index (RI)	1	1.5

Table 6.1: Composition of Bandwidth Class Scenarios (distribution, upload)

with SBR upload and 2*SBR download bandwidth; and 55% of POOR peers, with SBR/2 upload and 2*SBR download bandwidth. This amounts to a resource index of $RI_{HH-LB} = 1.045$.

The HH-LB scenario aims to show the system's behavior when bandwidth resources are scarce and asymmetrically distributed throughout the population. The total serving capacity is barely sufficient to provide every peer with a complete stream.

High Heterogeneity, High Bandwidth (HH-HB) This scenario encompasses four bandwidth classes: 4% of VERY RICH peers, with 10*SBR upload and 10*SBR download bandwidth; 20% of RICH peers, with 3*SBR upload and 3*SBR download bandwidth; 21% of NORMAL peers, with SBR upload and 2*SBR download bandwidth; and 55% of POOR peers, with SBR/2 upload and 2*SBR download bandwidth. This amounts to a resource index of $RI_{HH-HB} = 1.485$.

Here we noticeably increase the upload capacity of the two richest bandwidth classes. As a consequence, the total available bandwidth exceeds the minimum amount required for the complete stream distribution by nearly 50%. The resulting scenario aims to approximate the heterogeneous bandwidth distribution observed by recent studies [96][27] on resource availability in peer-to-peer file-sharing networks.

Low Heterogeneity, Low Bandwidth (LH-LB) This scenario encompasses two bandwidth classes: 20% of RICH peers, with 2*SBR upload and 2*SBR download bandwidth, and 80% of NORMAL peers, with SBR upload and 2*SBR download bandwidth. This amounts to a resource index of $RI_{LH-LB} = 1.2$.

The main challenge in this scenario is the small difference between the bandwidth capacity of the two classes of nodes, together with the presence of a relatively low excess of overall resources.

Low Heterogeneity, High Bandwidth (LH-HB) This scenario encompasses two bandwidth classes: 20% of RICH peers, with 4*SBR upload and 2*SBR download bandwidth, and 80% of

NORMAL peers, with SBR upload and 2*SBR download bandwidth. This amounts to a resource index of $RI_{LH-LB} = 1.6$.

This scenario is designed to examine the behavior of nodes in a two-class scenario where bandwidth resources are abundant.

Uniform Scenarios, High and Low Bandwidth These simple and self-explanatory scenarios are useful to provide results to support comparisons to structured systems. The low-bandwidth version (unif-LB) encompasses a single class of nodes with SBR upload and 2*SBR download bandwidth. The high-bandwidth version (unif-HB) encompasses a single class of nodes with 1.5*SBR upload and 2*SBR download bandwidth.

Churn Scenarios We experimented with few synthetic arrival and departure patterns, to introduce a way to evaluate the impact of user activity on the system behavior. We tried to approximate conditions that could be encountered in real-world environments, such as flash crowds and sudden departures. The arrival patterns we adopted are:

- ATONCE: all the nodes join the network at the instant $t = 0$
- SPIKE: 25% of the nodes join the system at $t = 0$; the remaining 75% come in together after the system reaches stability

The duration of the life of a node in the system is assigned in the following ways:

- NOLEAVE: nodes never leave the system until the end of the simulation
- SQUIT: sudden departure of 50% of the nodes

6.3 PULSE Parameters: How to Set Them?

When we launch a PulSIM run, we would like to just provide the arrival pattern and bandwidth distribution of the nodes. However, several protocol parameters have a decisive role both on the initialization and on the evolution of the system. It is necessary to correctly set these parameters before actual results can be obtained from the simulator.

Node initialization, for instance, is an especially sensitive phase, as new incoming nodes have an empty buffer and must execute the various buffer synchronization algorithms before they can operate normally. Especially when several nodes join in a small time frame, the odds that a node manages to fully initialize its buffer depends on many factors: the available excess resources are an important factor, but several buffer parameters (width of sliding windows, sliding tolerance, initial window position) also play a critical role. If a simulated system does not initially converge, then we will not be able to get any meaningful result from that trace.

Problems in the initial system convergence are caused by both the simulation artifacts and by the intrinsic properties of the algorithms. We remember that PULSE is a dynamic system based on positive feedback: this means that the initial conditions can have a decisive weight on the short-term stability of the system. In simulation, the stability problem is harder to address, since the "inertia" of simulated peers is lower than in the real world (because of the fixed propagation delay, oracle-like sources of knowledge, synchronized chunk transfer, etc.) and consequently the randomness of their actions is higher.

In the following pages, we explore a limited subset of the protocol parameter space by way of practical heuristics and trial-and-error. Our goals here are to begin to understand the fundamental system dynamics and to settle on a small range of possible protocol values that will give useful simulation outcomes.

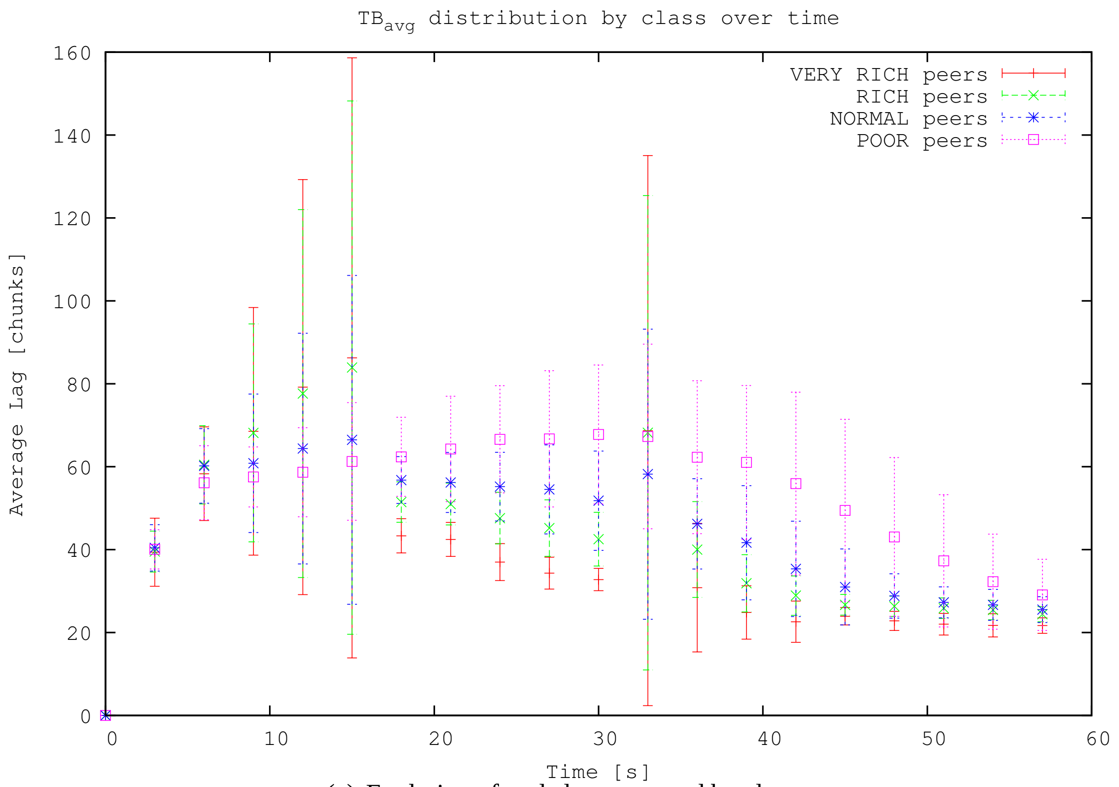
6.3.1 Initial Transitory Phase

The simulations always begin with several nodes joining the PULSE system: depending on the churn scenario, several hundreds of nodes can appear during the same time-step or within a short interval. Initially, joining nodes have an empty buffer, and must start to fill it by interacting with other nodes. We also remember that, until the buffer has been fully initialized, a node cannot start to advertise real average lag measurements about its buffer (Chapter 3, Algorithm 1).

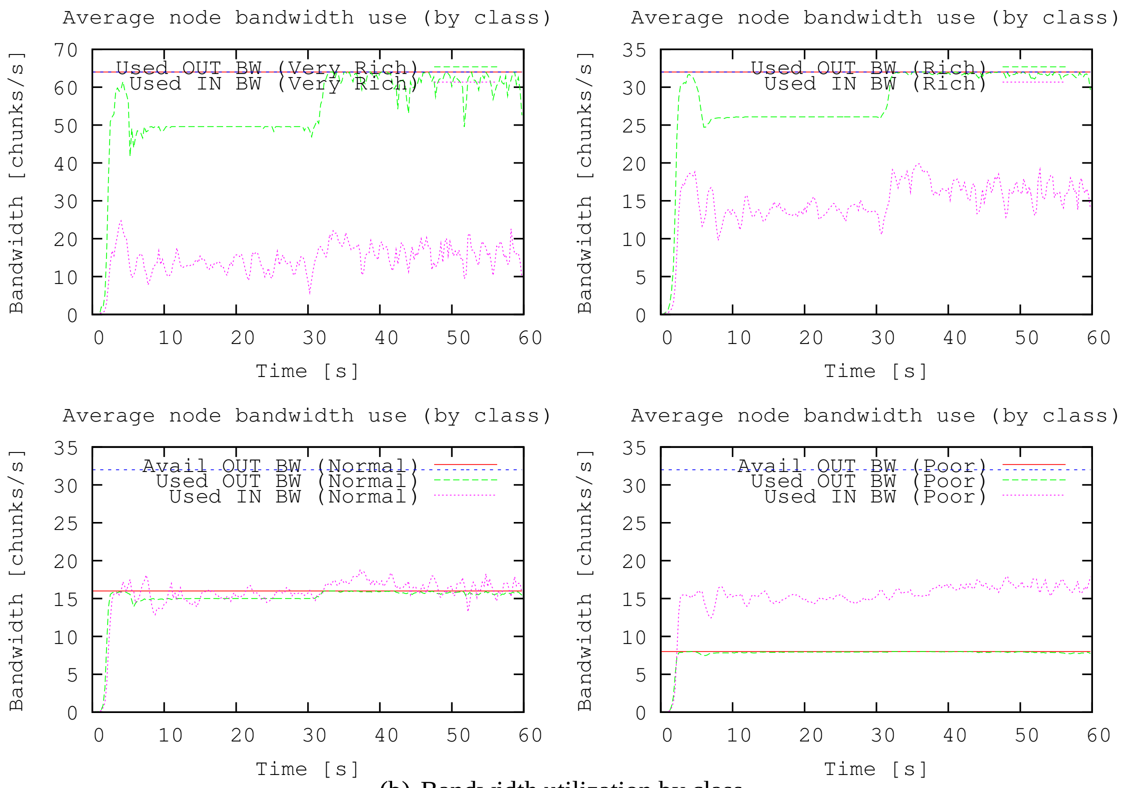
We recall that in the simulator the knowledge model allows an unrestricted access to the average node lag information. When they join, nodes do not have a valid node lag value, since their buffer is not initialized yet. As a consequence, all the nodes will randomly choose their partners during the first EPOCHs. Also, the widespread initial lack of chunks at the nodes at the beginning requires some time before node buffers begin to set a valid T_B and peer selection becomes effective.

The initial transitory phase should be of short duration (Figure 6.1): it is indeed common for most nodes to either initialize their buffers at the first attempt, or to retry the joining procedure once or twice before their buffer reaches the initialized state. The sudden increases in the variance of class lag, which are visible in Figure 6.1a at 12 and 33 seconds are due to several starved peers resetting their buffer as they reach the maximum allowed value of $T_B = T_D$. The bandwidth traces in Figure 6.1b confirm that, at $t = 33s$, all the peers are actively exchanging data. After $t = 33s$, we notice that the average download bandwidth of all the classes is *higher* than the stream rate, as the nodes reduce their lag from the source thanks to the presence of excess capacity. After the average system-wide node lag drops to a stable minimum, which approximates the mean delay required for the propagation of each chunk through the system, the class download rates become equal or slightly lower than SBR: this event marks the beginning of the steady state phase, where nodes reach an equilibrium and keep receiving data with a stable lag.

Interesting insights can be obtained by examining the transitory evolution of the relationship between node lag and delay of chunk reception (introduced in Chapter 5, Figure 5.3). In Figure 6.2 we can see a sequence of nine snapshots of an HH-LB system taken at regular intervals of



(a) Evolution of node lag, averaged by class



(b) Bandwidth utilization by class

Figure 6.1: Transitory of a PULSE System (1000 peers, HH-LB, $R = 16$ chunks/s, $TW = 64$)

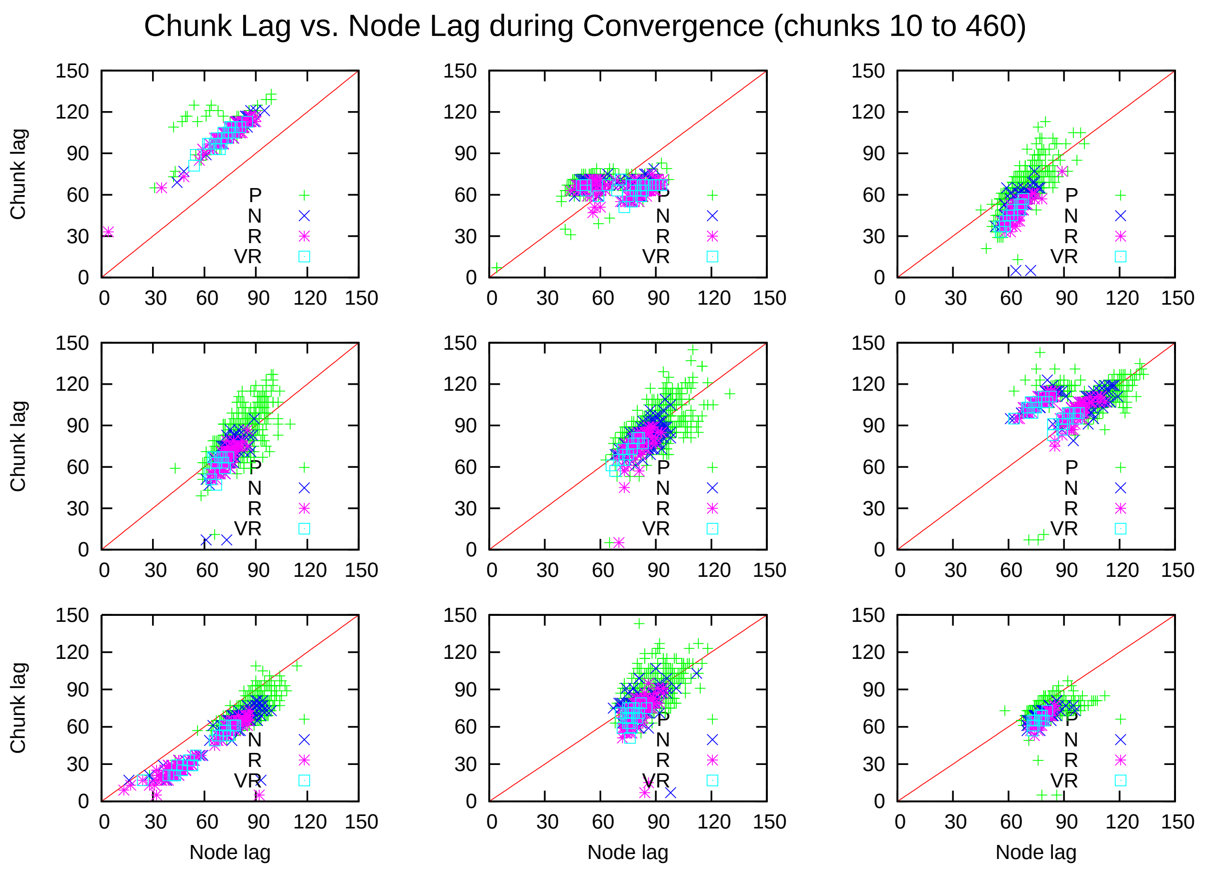


Figure 6.2: Snapshots of Chunk Reception Lag vs. Node Lag During Convergence

50 chunks during the first thirty seconds of convergence, ordered from top left to bottom right. The evolution of the "system cloud" shows the gradual and natural emergence of a group of resourceful nodes from the VR and R classes: they first coalesce in the midst of the other nodes (Figures 2 to 5); then, nodes that did not manage to connect on their first attempt rejoin the system after a buffer reset (Figure 6); because of the local excess of node capacity, the more resourceful nodes in the system slowly overrun the others, "gaining ground" and gradually reducing their node lag (Figure 7). We notice that, during the whole process, the most resourceful nodes are normally located below the diagonal line $y = x$, that is they receive chunks with a lower lag than their current node lag, while the poor nodes stand out as being spread along and above the diagonal. The two last Figures show the gradual stabilization of the system: once the resourceful nodes have aggregated at the front of the cloud, the rest of the system begins to collapse toward them, and the cloud reduces slowly but steadily its average global lag until it reaches a stable equilibrium around an average of 24 chunks as in the simulation above (Figure 6.1).

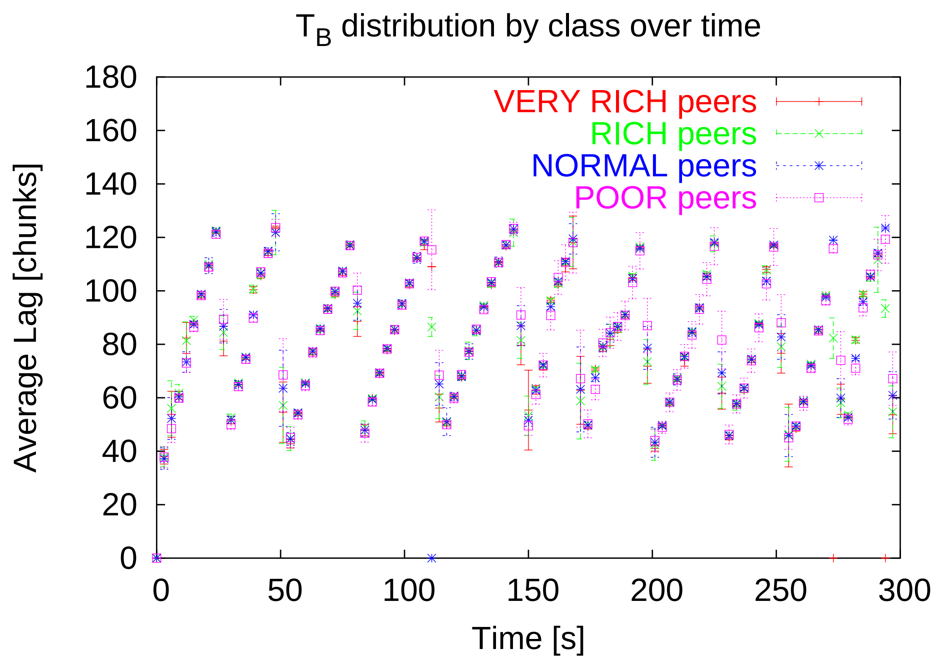


Figure 6.3: Typical Pattern of Instability due to a Small Trading Window ($TW = 32$)

6.3.2 Critical Parameters

We define **critical parameters** those system parameters that appear to have a strong non-linear effect on the initial outcomes of the simulation. These parameters affect both the initial chance of the system converging to a stable state, and can also determine long-term instability phenomena. The critical parameters are the Trading Window Size (TW) and its Window Sliding Tolerance (S). Our earliest experiences with the simulator have shown that these parameters are the most important factors during the initial transitory phase: if the Trading Windows are too short, nodes do not manage to obtain enough chunks in a timely manner and fail to develop a significant trading window overlap with their peers and the whole system may become stuck in the initialization phase. The second critical parameter is the loss tolerance of the sliding window: in general, the lower this value, the longer it takes for the nodes to begin a normal exchange activity, as trading windows tend to become stuck just after the initialization phase is completed.

The failure of the system to settle in a stable state gives as its outcome a characteristic node lag evolution: all the nodes in the system appear to reiterate their initialization attempts in a synchronous way, producing a periodical sawtooth plot (Figure 6.3). Interestingly, the frequency of the peaks appears to be inersely proportional to the width of the trading windows: under similar bandwidth scenarios, the larger the windows, the less frequent the massive re-initializations. Instability stops altogether once a threshold window size (which mainly depends on the bandwidth distribution scenario) is reached. For instance, for populations of up to 1000 nodes, instability appears to cease when the trading window is larger than 48 chunks. We experimented with values of TW ranging from 16 to 256 chunks: we noticed that, once a system is stabilized, further increasing the window size leads to higher average node lags and to a slower execution of the chunk selection algorithm (in addition to the larger size of control messages, a more practical concern).

The external variables that come into play in a simulation have an important role in determining the sensitivity of the initial transient phase to the two critical parameters above.

- The *capacity of the source* is especially critical with respect to the outcome of system initialization. We observed that, when the source bandwidth is less than twice the stream bitrate (for initial populations larger than 100 nodes), the transient phase either lasts longer or does not settle, resulting in an unstable system. Increasing the source bandwidth has a dramatic effect on system convergence: starting from three times the stream bitrate, the likelihood of experiencing initial convergence problems becomes much lower even when global capacity is scarce ($RI \sim 1$).
- A high level of heterogeneity coupled to a low resource index for a given scenario also results in increased chances that the initial transient phase will last for a longer time, as the initial connections among nodes are established at random: high asymmetry and low global capacity imply that, if several resource-rich nodes get a bad initial placement, the resources of the rest of the peers are not sufficient to sustain the system initialization.

We also noticed that the initial population size impacts the simulated system convergence. Even given a sufficient source capacity, we observed that the initial convergence tended to be slower or less likely for small initial populations (under 50). We attribute these convergence problems to the increased weight on the initial system behavior of random choices and concurrent actions by the simulated nodes.

6.3.3 Long-term System Stability

Arguably, a system that does not reach convergence at initialization could still manage to operate on the brink of instability: at the beginning, the upload capacity required by the system is higher than at steady state, as nodes have to fill their empty Trading Windows, and the upload bandwidth of each node is further limited by the lack of chunk diversity among the node buffers (content bottleneck).

We have however encountered several scenarios in which a system that initially converges does slowly diverge over time: this usually happens when the available upload capacity is scarce and uniformly distributed across the population. The lack of upload capacity in the system usually results in a subset of the nodes that start to increase their lag and eventually reach the *buffer reset* threshold. If this subset is small, the effect on the global system of few nodes resetting their buffer is negligible, and upon one or more re-connections these nodes may be able to reach a stable position. However, if this subset contains a large majority of the nodes, then the entire system may become unstable, with a periodical behavior that sees almost every node re-initialize its buffer at the same time (similar to Figure 6.3).

The critical parameter, in the case of long-term stability, is the total bandwidth capacity of the system. When resources are sufficiently available ($RI > 1$), the other parameters that come into play and can determine the speed of system convergence appear to be:

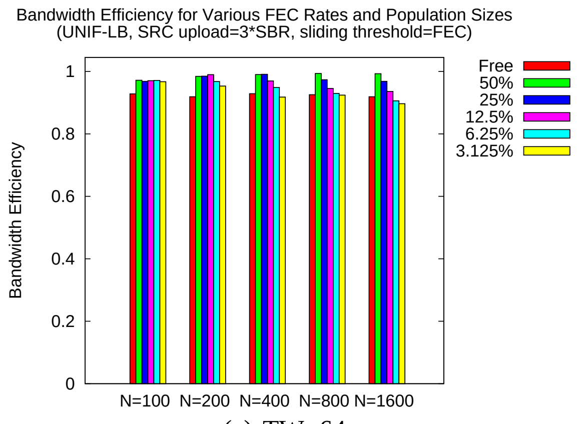
- Asymmetry of bandwidth scenario: counter-intuitively, the higher is the bandwidth asymmetry in a given scenario, the faster stability is reached. We will analyze this phenomenon in detail in the following pages.
- Trading Window Size (TW): while somewhat slowing the node initialization process, a larger TW gives the system an improved "inertia", since the content diversity between nodes becomes large and the probability of finding nodes with an overlapping window increases.

On the other hand, when resources are constrained ($RI \sim 1$), the Sliding Tolerance S emerges again as an additional critical factor: even if a system is 100% efficient, that is all the available upload capacity is exploited, the delays introduced by the random propagation of the chunks will hinder the progression of the sliding windows: if chunk losses are not allowed by the sliding window, and without an excess of bandwidth that could be used to recover the 'lost' chunks, the node lag is bound to increase and eventually reach the buffer reset threshold.

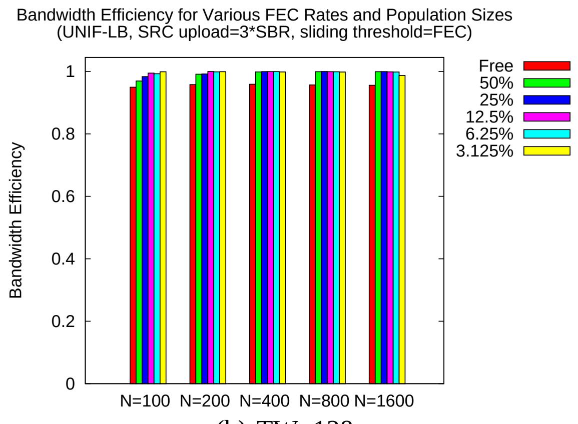
The use of an appropriate amount of FEC can improve the stability of the system, at the price of an effective reduction in the media bitrate: defining an appropriate amount of FEC that balances stability and efficiency is an interesting challenge. If we consider the worst-case scenario, an uniform capacity distribution with $RI = 1$ (UNIF-LB), we can observe the relationship between $\frac{S}{W}$ and the bandwidth efficiency ξ: in Figure 6.4 we plot the measured efficiency for simulated systems ranging from 100 to 1600 nodes using values of $\frac{S}{W}$ from 3.125% to 50%. For comparison, measurements have also been performed for an hypothetical case of $\frac{S}{W} = 100\%$, i.e. with no sliding constraints ('Free').

From Figure 6.4a, we notice that, the smaller the amount of FEC, the faster efficiency decreases with increasing system size. Efficiency is affected because, due to the fact that chunks have to traverse more nodes, buffer windows become less and less synchronized as the average lag increases. In turn, a lower efficiency further accelerates the growth of average node lag. Remembering the basic constraint of lossless media reception, which in the case $RI = 1$ requires an efficiency $\xi \geq (1 - \frac{S}{W})$, we can deduce from the observed efficiency whether a system of a given size may achieve or not a stable reception. For instance, a system with 3.125% FEC for a population of 200 nodes has a measured efficiency of 0.962, which is lower than the minimum efficiency required for stability[4]; a system with 6.25% FEC has $\xi = 0.951$ for a network of 400 nodes, which is more than the minimum, but becomes unstable at 800 nodes ($\xi = 0.9250 < 0.9375$).

[4]As throughput depends in turn on stability, once a system falls beyond the required efficiency threshold, the measurements become only indicative, as the system will no longer be in a stable state.



(a) TW=64



(b) TW=128

Figure 6.4: Impact of Sliding Tolerance on Efficiency with $RI = 1$

The use of larger Trading Windows improves stability and efficiency regardless from the amount of FEC: in Figure 6.4b we can remark that, with a TW length of 128 chunks (twice the size), the efficiency for all $\frac{S}{W}$ ratios is improved and stability starts to decay much later.

We can also observe that, when the window is *free* and slides without constraints, the efficiency for small population sizes is in many cases lower than when a sliding threshold is set, as an unspecified amount of chunks may have been lost as the window advances. However, we notice that the average efficiency of *free* windows is more or less stable under increasing system sizes ($\xi \sim 0.92$ for $TW = 64$, $\xi \sim 0.95$ for $TW = 128$), while in the case of constrained windows efficiency tends to drop as instability arises. This result confirms that PULSE is ill-suited for use in cooperative scenarios with $RI \sim 1$ and where node upload capacity is uniformly distributed: in these cases, any data-driven architecture that uses *(i)* a large Trading Window and *(ii)* a small amount of FEC[5] will obtain better scalability without introducing any stability concern.

6.4 Effects of the Peer Selection Algorithms

In this section, the macroscopic effects of the various parameters involved in peer selection are evaluated. We first experiment with the main system parameters, defining a protocol reference for comparing all subsequent results. Then, we investigate the role of the two peer selection mechanisms by inspecting the behavior of the metrics defined in Chapter 5 under a number of protocol variants.

[5]Larger amounts of FEC in this context can only increase the likelyhood of lossless media reception.

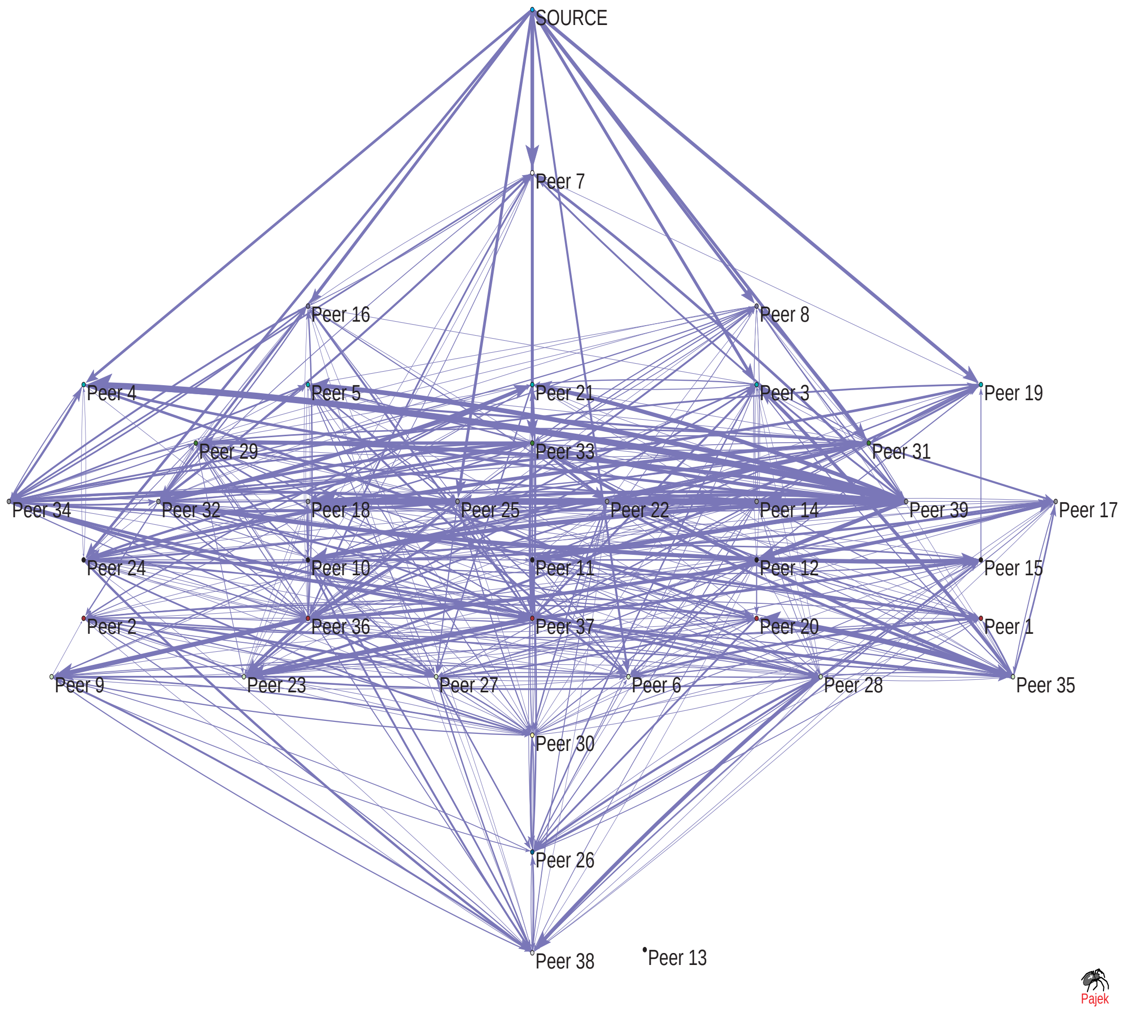


Figure 6.5: Snapshot of Data Exchanges during an EPOCH (HH-LB)

Figure 6.5 offers a sample snapshot of PULSE, taken during a small-scale simulation (40 nodes), visualizing all the data exchanges that have happened during an EPOCH period at steady state. In this picture, nodes are ordered from top to bottom by increasing node lag (the source is at the top of the graph). We can notice the prevalence (in number and volume) of horizontal connections, established between nodes with approximately the same lag.

6.4.1 Varying the Number of Connections

The parameters we define as standard to simulate[6] PULSE include a TW of 64 chunks (i.e. four seconds worth of data) with Sliding Window tolerance of $S = 25\%$, and an upper limit of 8 FORWARD connections and 4 MISSING connections. In Figure 6.6, which like Figure 6.1a represents the time evolution of the average class lag, we see that a system containing 1000 nodes under the HH-LB scenario manages to quickly stabilize around a small average lag value

[6]In all our simulations, the chunk rate is set to 16 chunks per second, that is 4 chunks per simulation step.

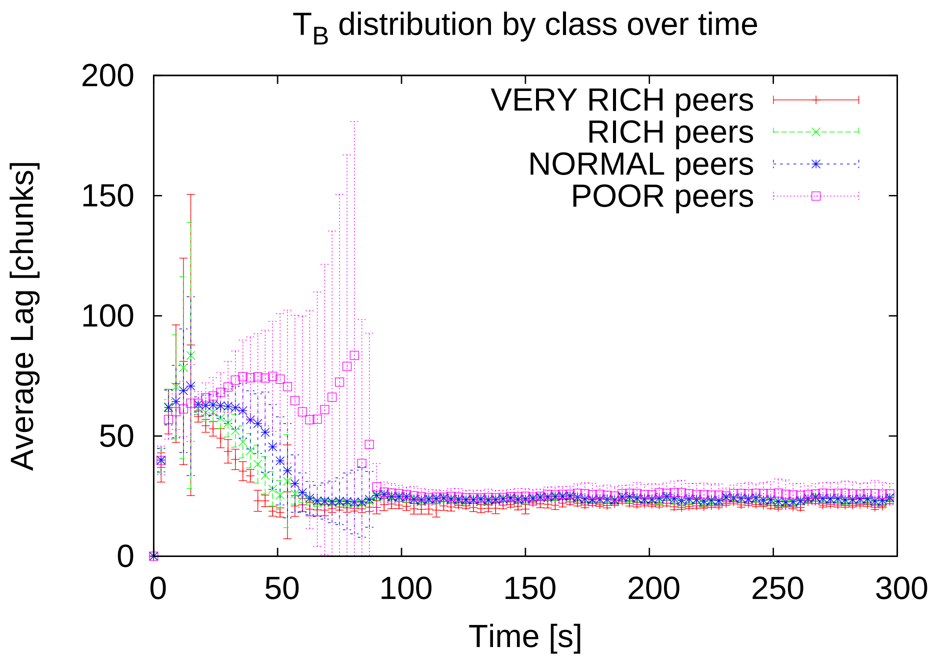


Figure 6.6: HH-LB Scenario: PULSE with Standard Parameters (4 M, 8 F, $TW = 64$)

	NO FWD				4 FWD				8 FWD			
Scenario	VR	R	N	P	VR	R	N	P	VR	R	N	P
HH-HB	0	0	23	181	0	0	0	0	0	0	0	0
HH-LB	13	53	59	242	0	0	4	183	0	0	0	0

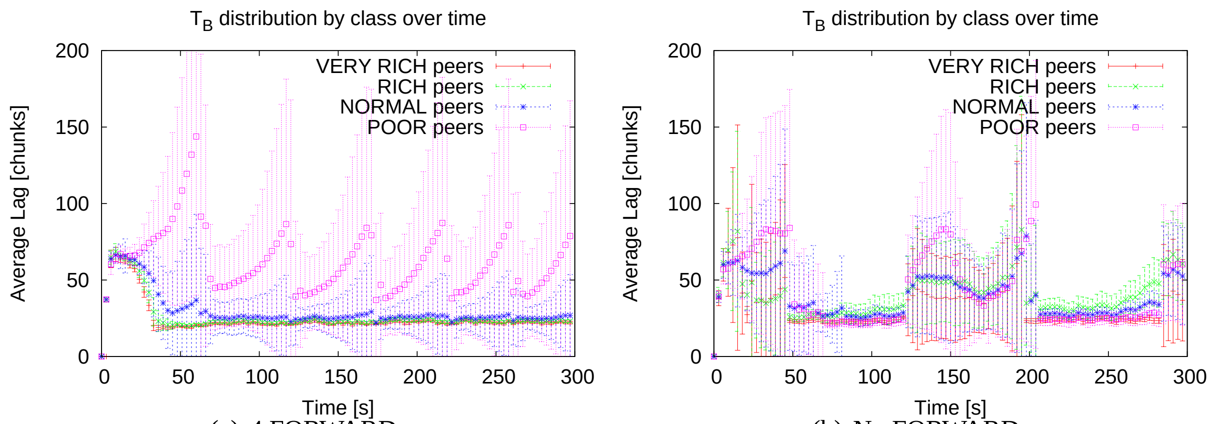
Table 6.2: Buffer Reset Statistics: Unstable Peers by Class at Steady State

of about 30 chunks (i.e. about 2 seconds) with a low, constant variance.

In Figures 6.7a and 6.7b we see how the removal of (respectively) four and eight FORWARD connections impacts the performance of the previous scenario. We notice how the presence of only a few FORWARD connections has a strong stabilizing effect on the whole system, greatly reducing the variance of lag for the three richest classes. In Figure 6.7a the poorest class is the only one that begins to suffer from starvation, with periodic re-connections of a small part of its nodes. It is interesting to notice that the repeating re-connections of these poor nodes do not affect significantly the performance of the remaining classes.

When we altogether remove all the FORWARD connections,we can notice how convergence still takes place, but the overall system stability is much weaker: all the classes are visibly disturbed by the fact that buffers are frequently reset (e.g. massive reconnection at $t = 120s$, leading to a massive disconnection at $t = 200s$) and many nodes either have an unstable lag values or become stuck in the initialization phase (Table 6.2).

There are several common factors that emerge from the pictures above:



(a) 4 FORWARD (b) No FORWARD

Figure 6.7: HH-LB Scenario: Reducing the Number of FORWARD Connections

- The different bandwidth classes appear to settle at different values of average lag, with richer classes being nearer to the source than poorer ones. This is in agreement with our intuition that the coordinated effect of incentive-based (**tit-for-tat**) and performance-based (**lag-based feedback**) peer selection would allow the generation of clusters of peers with similar resource availability.

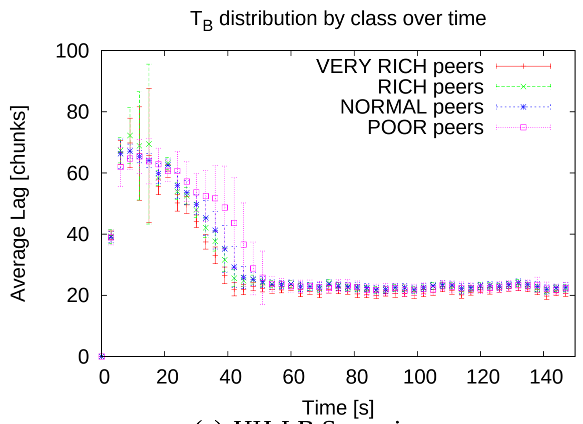
- The initialization of the simulated system is relatively quick, even despite the lack of a significant amount of excess bandwidth and the limits of the knowledge model used by the simulator. In the three plots, which are typical realizations of the system behavior in this scenario, system convergence is achieved between 50 and 100 simulated seconds, that is about 25-50 rounds of peer selection.

- Richer classes achieve their convergence before poorer ones, while the poorest class suffers the most from global bandwidth scarcity.

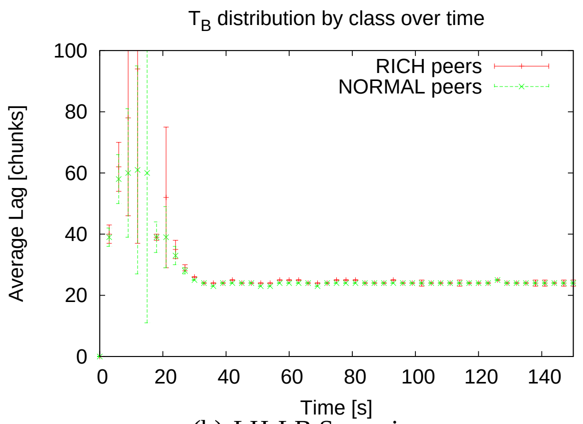
6.4.2 Lag Performance across Bandwidth Scenarios

In Figure 6.8 we present four realizations of simulated PULSE system under the bandwidth scenarios presented above. In these plots we can appreciate several qualitative properties of the PULSE system, which confirm and integrate our previous remarks:

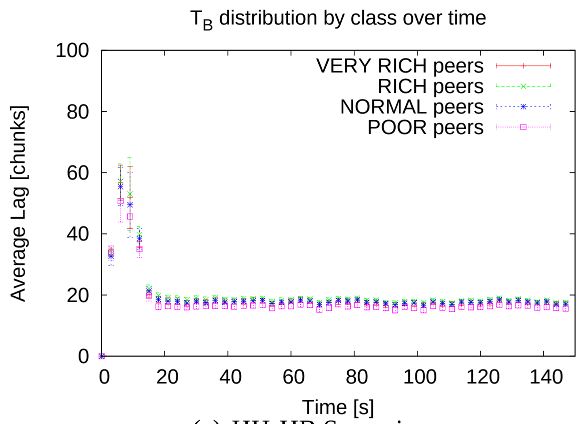
- *The RI of a scenario influences convergence speed*: comparing the HH-LB and HH-HB scenarios, the time required for nodes to reach a steady position in the system is much lower when excess bandwidth is abundant ($< 20s$ for $RI_{HH-HB} = 1.485$) than when it is scarce ($\sim 60s$ for $RI_{HH-HB} = 1.045$). An analogous observation can be made to a lesser extent for the LH-LB and LH-HB scenarios.



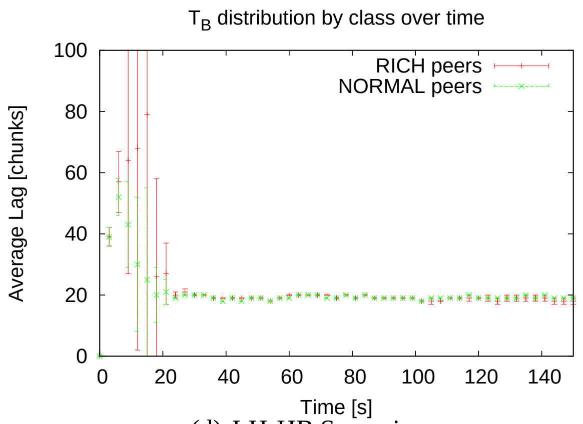
(a) HH-LB Scenario



(b) LH-LB Scenario



(c) HH-HB Scenario



(d) LH-HB Scenario

Figure 6.8: Examples of System Evolution in Various Bandwidth Scenarios

- *The RI of a scenario influences the average system lag*: the availability of excess resources reduces the average node lag. Average lag ranges from about 24 chunks for the HH-LB and LH-LB scenarios to about 18 for HH-HB and LH-HB scenarios.
- *The presence of bandwidth heterogeneity reduces the average system lag*: comparing the HH-LB and LH-LB scenarios, we see that they settle on a similar average lag, despite the fact that the RI for the former is much lower than the latter (1.045 vs. 1.2).
- *Upload capacity influences the order of average class lag values*: it is possible to see that the average class lag increases as the upload availability decreases in the HH-LB scenario.

6.4.3 Understanding Node Interactions

To shed more light over the global behavior of a running PULSE system, we are going to apply the behavioral metrics presented in Section 5.3. We hereby recall the definition of these metrics, namely Class Affinity $\Phi(\alpha, \beta)$ and Class Friendliness $\Psi(\alpha, \beta)$:

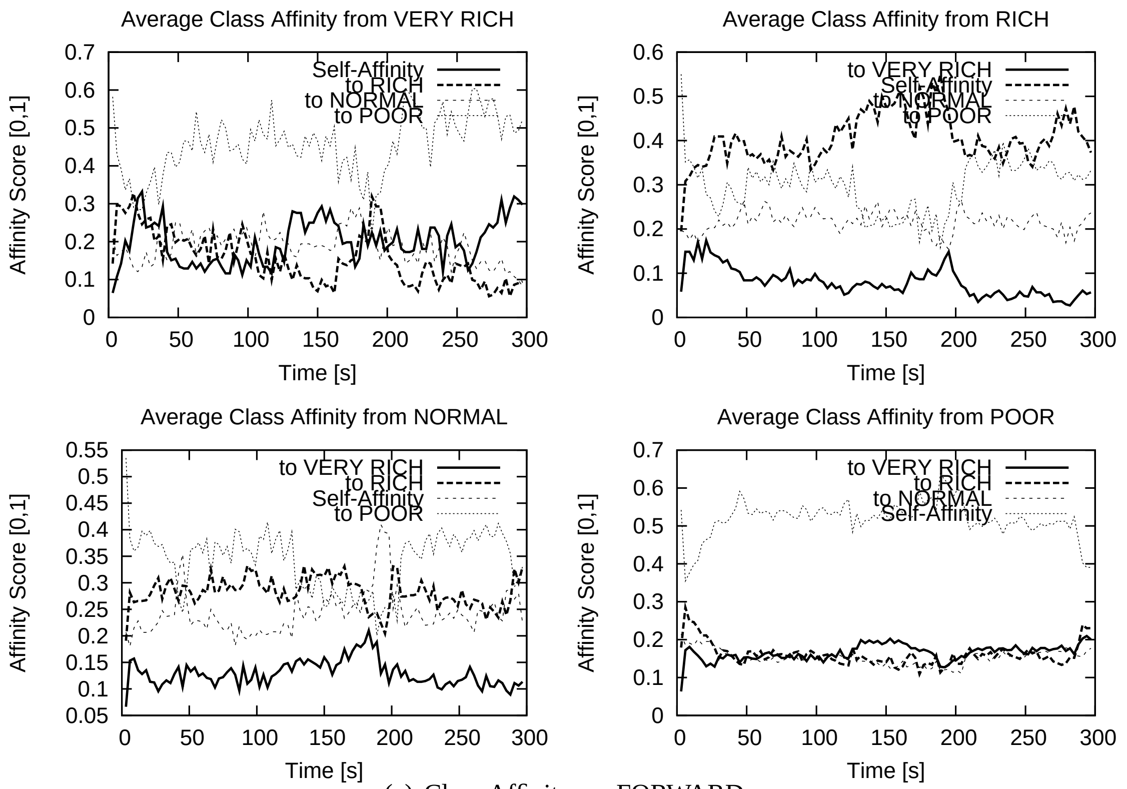
$$\Phi(\alpha, \beta)(t) = \frac{\sum_{n\in\alpha} \|n\text{'s MISSING links toward nodes of class } \beta\|}{\sum_{n\in\alpha} \|n\text{'s total MISSING links}\|} \tag{6.1}$$

$$\Psi(\alpha, \beta)(t) = \frac{\sum_{n\in\alpha} \|n\text{'s active FORWARD links toward nodes of class } \beta\|}{\sum_{n\in\alpha} \|n\text{'s total active FORWARD links}\|} \tag{6.2}$$

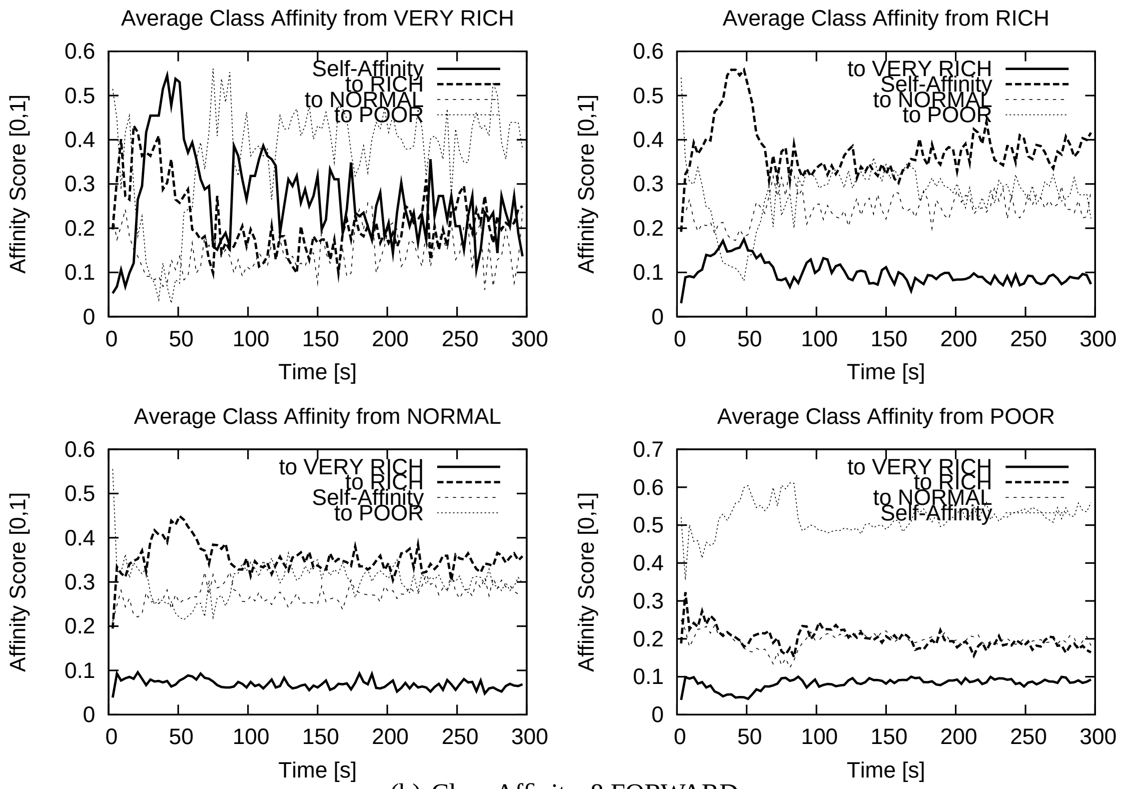
Affinity and Friendliness With and Without Altruism We compute Affinity and Friendliness values for the previous simulation scenarios and provide a sample of the results in Fig. 6.9. We first observe the evolution over time of Class Affinity in the HH-LB scenario with a) no FORWARD connections and b) 8 FORWARD connections. The plots suggest that there is a correlation between the system's convergence status and the value of the Affinity metric: when instability is present, as in Fig. 6.9a, the Affinity values tend to widely fluctuate, especially when node reconnections take place. Looking more closely, we can clearly see that especially the self-affinity of the resourceful classes becomes higher during instability, suggesting that tit-for-tat plays a critical role in the initialization phase and whenever global shortage appears, but somewhat loses its relevance when the system operates without constrained resources.

Comparing Figure 6.9a and 6.9b, we see that only the Affinity values for the two richest classes show meaningful quantitative differences during the convergence phase. We also remark that the poorest classes show similar Affinity scores across the two scenarios. We can notice that self-Affinity for the richest class is initially much higher when FORWARD connections are allowed. This is a side effect of the interaction between altruism and tit-for-tat selection, which improves the relationships among peers with extra resources. In fact, data contributed over the FORWARD connections is also taken into account for the tit-for-tat selection at the receiver. This increases the likelihood that the receiver will want to react and establish a MISSING connection on the following EPOCH. As richer peers have more spare resources, they gain more MISSING relationships over time. Self-Friendliness (not shown) is also very high, as targets preferred for FORWARD links by the richest classes are mainly peers from the more resourceful classes.

Affinity and Friendliness in Different Scenarios We performed a full comparison of the average steady state values of Affinity and Friendliness we obtained from simulations of HH-LB and HH-HB scenarios where nodes are allowed to establish up to eight FORWARD connections (Table 6.3). Again, we did observe meaningful differences especially in the Affinity values between Very Rich (VR) and Poor (P) peers: in HH-LB the self-Affinity of the VR class is much higher in the low bandwidth scenario, while the affinity between P and VR is much lower. Relationships between the other classes do not seem to be affected by the presence of FORWARD connections: MISSING connections are established by richer classes toward poorer ones more or less with the same probability in both scenarios. Moreover, the high Affinity value between P and VR peers in the high-bandwidth scenario is due to the fact that poor peers receive from the



(a) Class Affinity, no FORWARD



(b) Class Affinity, 8 FORWARD

Figure 6.9: HH-LB Scenario: Class Affinity vs. Number of FORWARD Connections

Normalized Affinity								
	HH-LB				HH-HB			
	VR	R	N	P	VR	R	N	P
VR	*6.70*	1.12	**0.63**	0.77	*3.77*	1.33	**1.39**	0.55
R	2.56	**1.93**	1.06	0.53	2.74	**2.48**	1.57	0.17
N	**1.95**	1.77	1.19	0.56	**3.71**	2.00	1.40	0.32
P	**2.67**	0.93	0.81	1.01	**7.77**	0.63	0.35	0.82

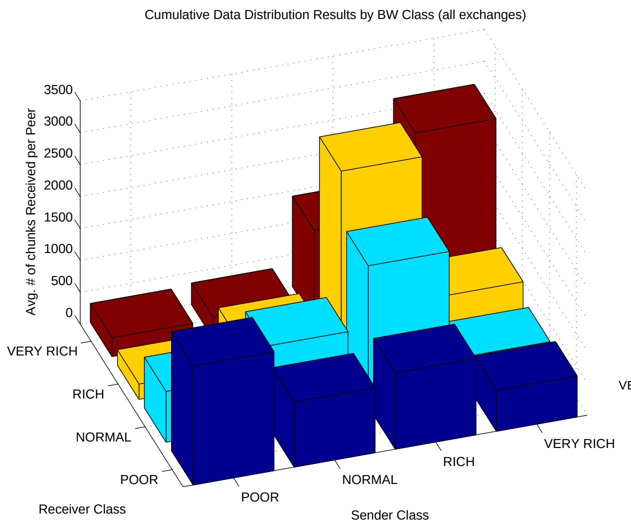
Normalized Friendliness								
	HH-LB				HH-HB			
	VR	R	N	P	VR	R	N	P
VR	*11.18*	**2.15**	0.47	*0.18*	*5.55*	**3.20**	0.51	*0.03*
R	**1.43**	**2.71**	1.10	*0.29*	**2.13**	**3.68**	0.68	*0.02*
N	**0.56**	**0.62**	1.43	*0.60*	**1.69**	**2.75**	1.63	*0.12*
P	**0.58**	0.60	0.83	1.24	**1.39**	0.80	1.01	1.03

Table 6.3: Comparison of Normalized Affinity and Friendliness
Comparison between HH-LB and HH-HB Scenarios with 8 FORWARD connections. The values highlighted in bold are significantly higher for HH-HB. The values highlighted in italic are significantly higher for HH-LB.

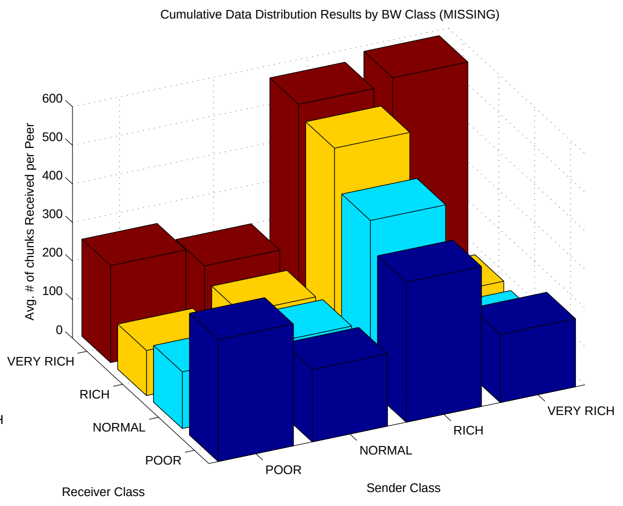
richest class more data than before in return for each successful exchange, and thus tend to reciprocate to them with higher probability. We conclude on these observations that the FORWARD exchanges do no interfere on the outcomes of the tit-for-tat selection, but help the richest nodes to fully exploit their bandwidth potential.

Friendliness results show another interesting trend: in both scenarios, the self-Friendliness value for each class is the highest on each row (except in a single case - N peers, HH-HB Scenario), meaning that nodes with a similar level of contribution tend to help each other out. The differences between the two scenarios can be explained by the different values of average class lag on which the nodes settle: while in the LB scenario the nodes are rather clustered around different lag values, in the HB scenario nodes from different classes are all mixed up in a smaller lag interval.

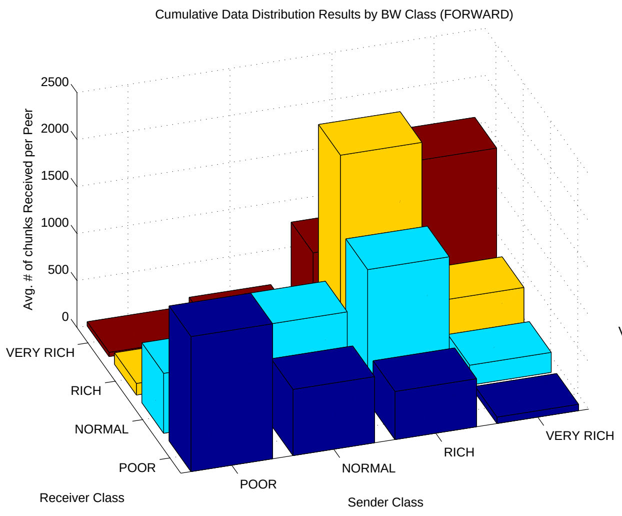
Data Weight of Relationship Types We investigated Class Affinity and Friendliness, which describe the likelihood of the establishment of a MISSING or FORWARD connection between nodes that belong to two bandwidth classes. Since in a data-driven system the existence of a connection does not guarantee that it will be actually used to exchange data, nor how many chunks will pass during an EPOCH over that connection, we are now going to observe how much data is traded between pairs of bandwidth classes, and what are the types of connections that convey the largest amount of data. In Figure 6.10a we draw the cumulative amount of data chunks exchanged between the four bandwidth classes during a five-minute HH-LB simulation, normalized by the



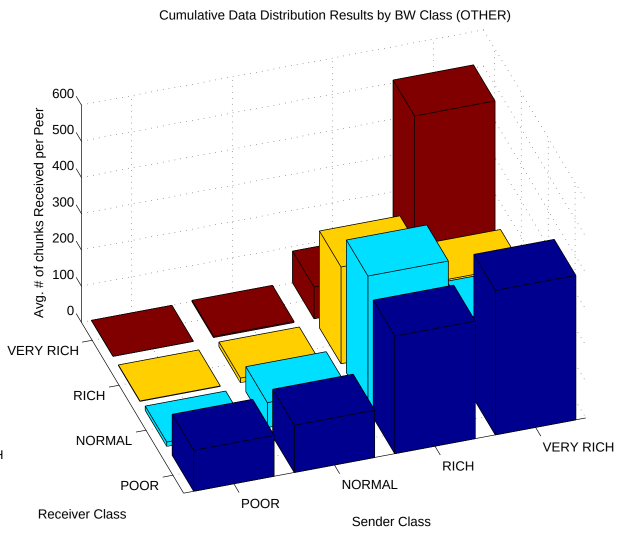
(a) Total Exchanges



(b) Only MISSING



(c) Only FORWARD



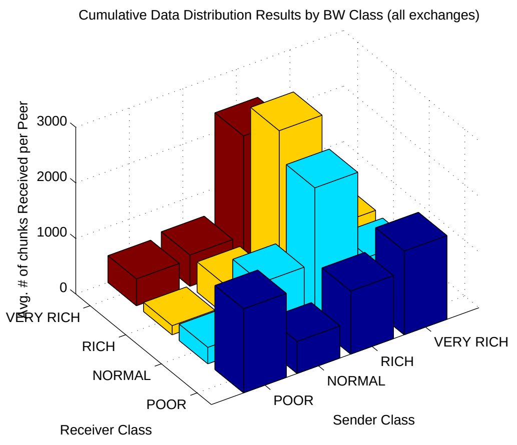
(d) Only NEW

Figure 6.10: Weight of Data Exchanges over Different Connections (HH-LB Scenario, 8F, 4M)

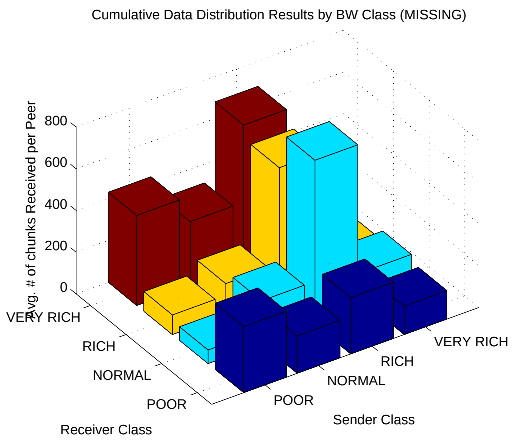
size of the receiving class[7]. We can remark that the VR and R classes provide to themselves the majority of the data, while receiving very little from the other classes. Conversely, the N class relies heavily on the contribution of the R class, while the P class also receives an important amount of data from both R and N classes.

When we observe the amount of data exchanged over the MISSING connections alone (Figure

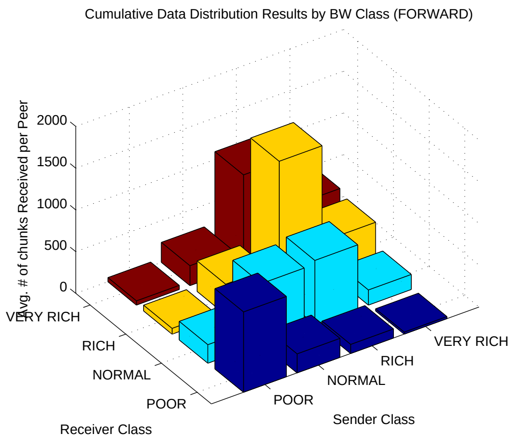
[7]Nodes can receive at most the total number of chunks that have been distributed by the source (SBR*t, in this case 16*300=4800). Hence, in the "total exchanges" histograms, the sum along the x (receiver class) axis is equal to the average number of chunks received by a class during the simulation, which is roughly the same for every class.



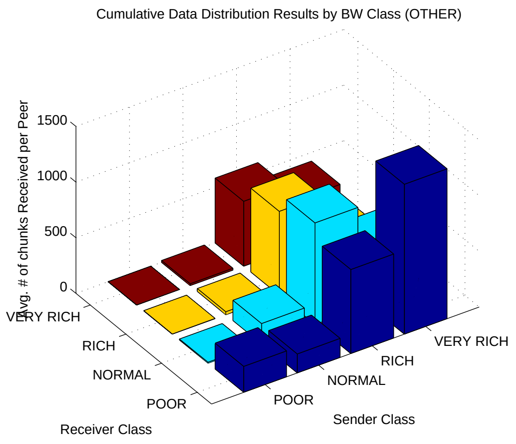
(a) Total Exchanges



(b) Only MISSING



(c) Only FORWARD



(d) Only NEW

Figure 6.11: Weight of Data Exchanges over Different Connections (HH-HB Scenario)

6.10b), we first notice that they convey only a small part of the stream data to the two richest classes (35% for the VR class, 18% for R). Then, we can observe how reciprocation is not exactly proportional to the amount of node capacity: for instance, the contribution by the R class is not reciprocated evenly neither by the N peers nor by the VR. Looking at the FORWARD connections (Figure 6.10c), we see that their role in data exchanges is quantitatively preponderant. Surprisingly, we observe a high degree of symmetry in the amount of data exchanged using FORWARD connections, with peaks in the self-contribution for most classes. These results confirm the fact that altruism, when applied on a system which has been loosely organized by the repeated action of tit-for-tat peer selection, is useful as it helps consolidate the relationships between the more resourceful nodes. Finally, the NEW connections (Figure 6.10d) contribute a small amount of data: it is interesting to observe that a significant 10% of the stream is exchanged inside the VR class, as this is probably due to retributions to optimistic selections in the MISSING connections.

In Figure 6.11, we conduct an analogous analysis on a HH-HB simulation trace. The most striking observation that emerges from these results is the loss of relevance in the system economy of the VR class, despite its very high upload capacity (10*SBR): this fact suggests that the upload capacity of the VR nodes is not fully utilized. The R class, whose upload is now 3*SBR, becomes the most important provider of data for all the classes. Because of the widespread presence of excess resources, the MISSING mechanism loses much of its importance: in Figure 6.11b, we see that VR nodes do not send many chunks over MISSING connections, while R nodes increase their contribution compared to HH-LB. The FORWARD exchanges (Figure 6.11c) maintain their quantitative importance and the degree of symmetry observed above, except for the role of VR nodes, which receive from R peers much more than they give. Finally, we observe that the weight of NEW exchanges (Figure 6.11d) is higher than in the previous case: the extremely large amount of chunks sent by VR nodes (which have no longer a lag substantially lower than R, N, and P) to N and P nodes can justify for the higher amount of data sent by N and P nodes to VR nodes using MISSING connections (in Figure 6.11b).

Soft Fairness between Classes Soft Fairness $\mathcal{F}(\alpha,\beta)$ between classes α and β (with upload capacity $U_\alpha < U_\beta$) was defined in Section 5.3 as

$$\mathcal{F}(\alpha,\beta)(t) = \frac{\sum_{n\in\alpha}\sum_{m\in\beta} \mathbb{I}(\,T_B(n)(t) \leq T_B(m)(t)\,)}{\|\alpha\|\;\|\beta\|} \tag{6.3}$$

while, when $U_\alpha < U_\beta$, its definition becomes:

$$\bar{\mathcal{F}}(\alpha,\beta)(t) = \frac{\sum_{n\in\alpha}\sum_{m\in\beta} \mathbb{I}(\,T_B(n)(t) \geq T_B(m)(t)\,)}{\|\alpha\|\;\|\beta\|} \tag{6.4}$$

In Table 6.4 we compare the Soft Fairness results from the HH-LB scenario, sampled at steady state every 3 seconds and averaged over the last 90 seconds, with those from an HH-HB simulation, with and without FORWARD connections. The differences in the Soft Fairness values between the two scenarios with FORWARD connections allowed are quite impressive: we see that, under global bandwidth excess, the Soft Fairness of the two poorer classes is very low with respect to all other classes. This means that, more often than not, NORMAL and POOR nodes obtain a slightly lower lag than their richer counterparts.

This observation quantitatively confirms that, when resources are abundant, the tit-for-tat incentive mechanism becomes less relevant and is preempted by the altruism present in the peer selection algorithms. To further investigate the role of altruism under excess of resources, we turn to the results obtained when FORWARD connections are disabled. We notice that there is a much smaller deviation in Soft Fairness values between the two scenarios. We believe that this is due to the lack of FORWARD connections, the primary altruistic mechanism. Also, some degree of unfairness is present in both cases, for all classes except VERY RICH, probably as a consequence of the altruistic discovery mechanism used by MISSING peer selection. Finally, we can appreciate how the presence of FORWARD connections enhances the clustering effect of the

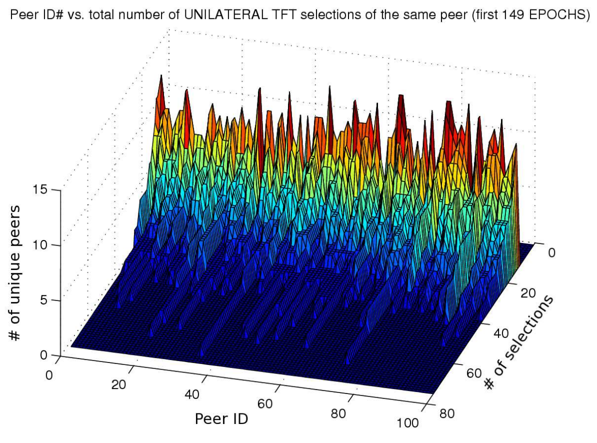
Soft Fairness between Different Bandwidth Classes								
	HH-LB, 8 FWD				HH-HB, 8 FWD			
	VR	R	N	P	VR	R	N	P
VR	=	**0.81**	**0.81**	**0.80**	=	0.75	0.66	0.36
R	0.63	=	0.65	0.71	0.55	=	0.51	0.23
N	0.69	0.53	=	0.65	0.46	0.30	=	0.30
P	**0.74**	**0.64**	**0.58**	=	0.21	0.12	0.16	=
	HH-LB, NO FWD				HH-HB, NO FWD			
	VR	R	N	P	VR	R	N	P
VR	=	**0.82**	**0.76**	**0.64**	=	0.86	0.67	0.53
R	0.76	=	0.42	0.41	0.81	=	0.25	0.35
N	0.68	0.34	=	0.48	0.56	0.20	=	0.45
P	**0.55**	**0.32**	**0.38**	=	0.46	0.34	0.38	=

Table 6.4: Soft Fairness: Comparing HH-LB and HH-HB Scenarios with and without FORWARD

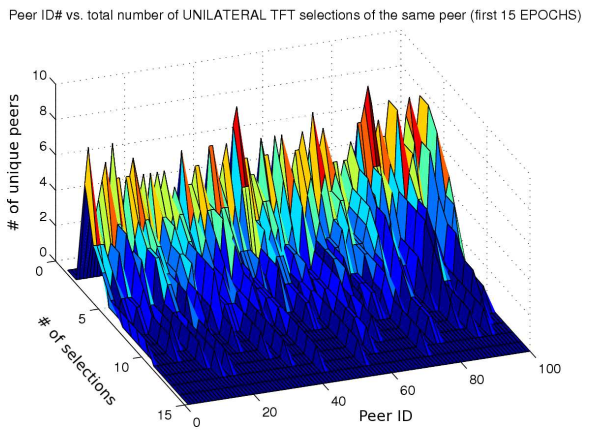
tit-for-tat incentive, especially when the total available bandwidth is scarce (highlighted in bold in Table 6.4).

Impact of Repeated MISSING Choices Finally, we examine the outcomes of the TFT-based peer selection algorithm. Our goal here is to obtain more detailed insights about the role of the two types of MISSING connections as the internal conditions of the system evolve over time. Figure 6.12 shows the cumulative duration of the MISSING interactions[8] between nodes in HH-LB scenario (P to VR from 1 to 100) over a time span of 300 seconds (150 EPOCHs) and 30 seconds (15 EPOCHs), differentiating the MISSING relationships between *unilateral* (established toward a partner which is not currently reciprocating) and *bilateral* (both nodes have a link open toward the other) connections. We can see in Figure 6.12a that nodes from all classes maintain few long-term relationships (peers selected more than forty times over 150 EPOCHs are rare), with a strong prevalence of one- or two-time node selections. If we concentrate our attention to the convergence phase (Figure 6.12b), we can notice a slight advantage for the richer classes in the number of medium-length interactions. Deeper insights can be obtained from the analysis of bilateral relationships: in Figure 6.12c we see that bilateral relationships are established in prevalence by the richest classes (VR and R). Moreover, when comparing the amount and duration of long bilateral relationships in 300 and 30 seconds (Figure 6.12d) we can see that they are mostly unchanged for the VR and R classes, while almost all the bilateral relationships among P nodes occur after the system has reached steady state.

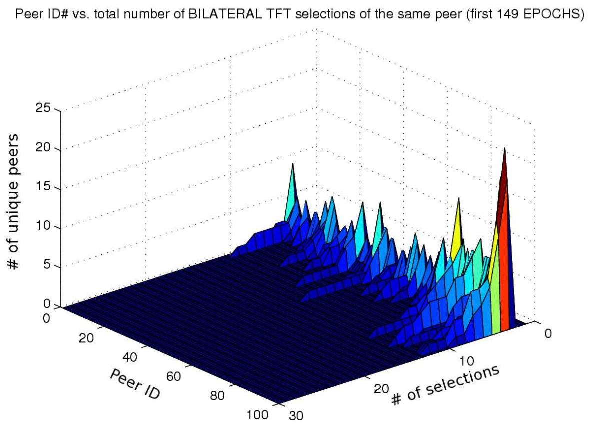
[8]To reduce the vertical scale of these plots, all the relationships that last less than ten EPOCHs have been purged from Figure 6.12a, and those less than two EPOCHs from Figures 6.12b, c and d.



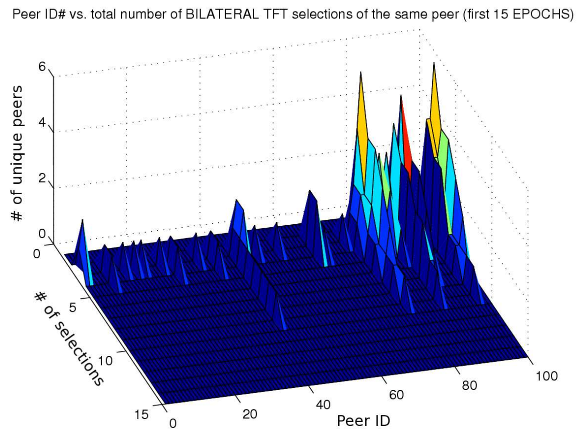
(a) Unilateral - First 300 seconds



(b) Unilateral - First 30 seconds



(c) Bilateral - First 300 seconds

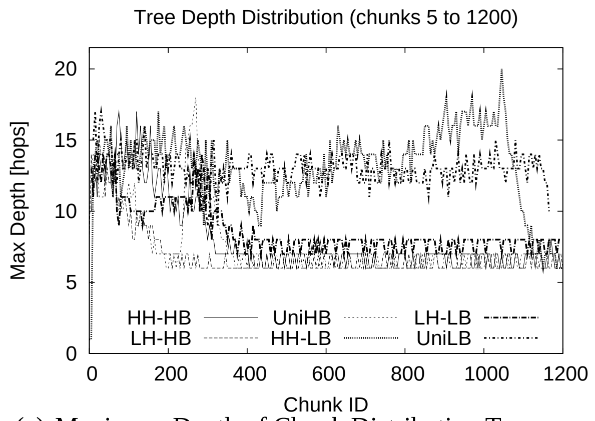


(d) Bilateral - First 30 seconds

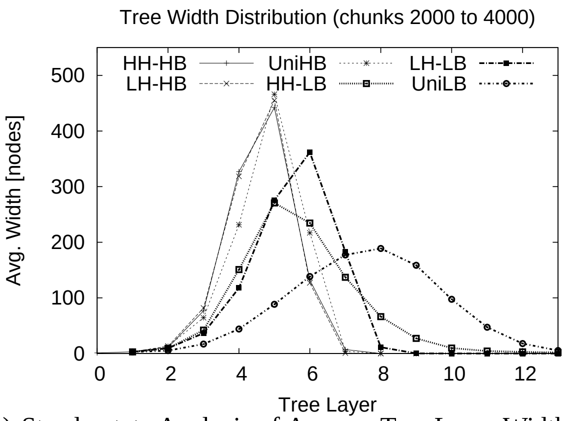
Figure 6.12: Total Duration of Unilateral/Bilateral MISSING Interactions (100 nodes, HH-LB)
In this figure Peer IDs are assigned to bandwidth classes in the following way: POOR from 1 to 54, NORMAL from 55 to 76, RICH from 77 to 96, VERY RICH from 97 to 100.

6.5 PULSE: a Quantitative Analysis

This section presents a comprehensive set of results about the performance of the standard PULSE algorithms and parameters. We first describe the data distribution performance from the point of view of the average properties of the data paths. Then, we observe the average placement of the nodes in the steady-state chunk distribution trees. Finally, we investigate the asymptotic scalability of PULSE up to medium-scale node populations to confirm our conjecture that the random distribution paths actually scale as trees, with a logarithmic dependence on system size. Finally, we observe the average composition of the distribution tree layers in term of node bandwidth class.



(a) Maximum Depth of Chunk Distribution Trees



(b) Steady-state Analysis of Average Tree Layer Width

Figure 6.13: Comparison of Max Depths and Average Widths of Distribution Trees (1000 nodes)

6.5.1 Analysis of Data Distribution Performance

Even if PULSE is a mesh-based system, the path that each data chunk follows on the overlay mesh is a tree. Trees will typically differ from chunk to chunk, depending on the current organization of the overlay: as the overlay connections are continuously renegotiated by each node in an independent way, one could expect that the properties of the different trees will vary a lot across different chunks.

Actually, this is not the case: Figure 6.13a shows the maximum tree depth for the first thousand chunks. It can be easily noted that subsequent trees have similar depth, and - more importantly - that tree depths tend to decrease over time, settling around an asymptotic minimum value. We can explain this observation with the aggregation process among resourceful nodes, as described in the previous pages. Thanks to the effects of the incentive-based peer selection, nodes with excess bandwidth manage to get data earlier than poorer nodes. The presence of altruism speeds up the data distribution process, as it increases the amount of data that rich nodes will exchange.

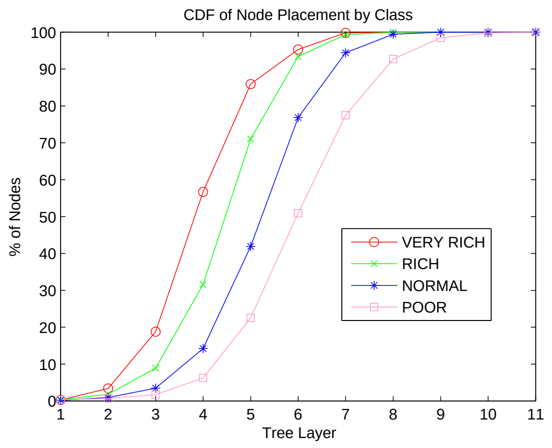
We observe that, according to our expectations, the trees from HB scenarios do converge faster, are usually shorter, and have top layers that are wider on average than those from the HH and LB scenarios. The scenarios that haven't yet reached convergence toward the end of the 75 seconds of the simulation above are HH-LB and Unif-LB: this fact confirms the importance of bandwidth excess during system initialization.

We recall from Chapter 3 that the stream source tends to give chunks to nodes with lower lag values. Hence, the rich nodes will be either directly served by the source, or receive recent chunks after few hops: in both cases, the likelihood that resourceful nodes will be placed close to the root in subsequent chunk distribution trees is high. We expect that this node placement will originate trees which are initially larger, and thus in average shorter than balanced trees with fixed degree. This is confirmed by Figure 6.13b: we can see that the HH-LB scenario generates trees with much better properties than the Uniform-LB scenario, despite the small RI difference.

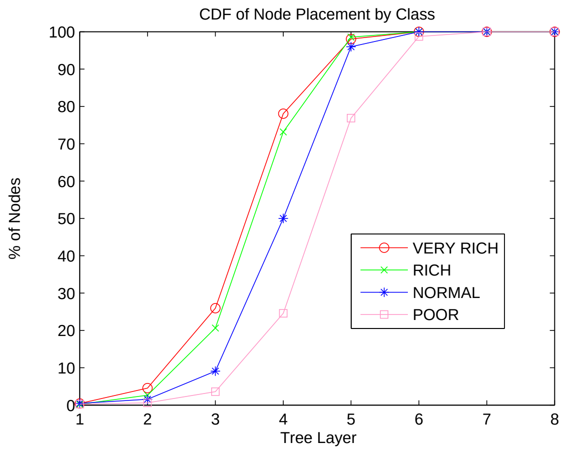
Max. Tree Depth / RI	HH	LH	UNIF
HB	6.30 (2.04) / 1.488	6.15 (1.59) / 1.603	6.30 (0.47) / 1.5
LB	10.10 (5.07) / 1.048	7.75 (2.78) / 1.203	11,10 (4,44) / 1.0

Avg. Tree Depth / RI	HH	LH	UNIF
HB	4.62 (0.31) / 1.488	4.59 (0.27)/ 1.603	4.81 (0.25) / 1.5
LB	5.67 (0.56) / 1.048	5.53 (0.23) / 1.203	7.57 (0.64) / 1.0

Table 6.5: Average (Std. Dev.) of Max. and Average Tree Depth at Steady State (1000 nodes)



(a) HH-LB Scenario



(b) HH-HB Scenario

Figure 6.14: CDF of Average Node Class Distribution at Steady State (1000 nodes)

Moreover, the average width of the first five tree layers in HH-LB and LH-LB scenarios is also very similar, despite the large RI difference: on the other hand, the tail of the HH-LB tree is on average few layers deeper.

Tree Depth at Steady State We can also notice how, once convergence is reached, the maximum and average tree depth remains largely stable: in Table 6.5 we show the average and standard deviation of both maximum and average tree depth (sampled every three seconds during one minute). As we expected, scenarios with higher RI have shorter maximum path lengths, while bandwidth heterogeneity only seems to slightly increase the variance of maximum tree depth. On the other hand, the presence of heterogeneity has a remarkable effect in reducing the average depth of distribution trees: this is particularly evident when comparing HH-LB to uniform-LB scenarios, as their RI values are close, yet the average path length is much lower in presence of a heterogeneous bandwidth distribution.

Node Class Distribution across Layers Figure 6.14 shows the cumulative distribution function (CDF) of the average node placement in chunk distribution trees for HH-LB and HH-HB

HH-LB	L1	L2	L3	L4	L5	L6	L7
VR	0.027	0.202	0.210	0.150	0.066	0.019	0.007
R	0.157	0.343	0.397	0.406	0.344	0.181	0.057
N	0.189	0.212	0.196	0.200	0.237	0.265	0.176
P	0.626	0.242	0.195	0.242	0.351	0.533	0.760

HH-HB	L1	L2	L3	L4	L5	L6	L7
VR	0.037	0.213	0.108	0.073	0.018	0.006	0.003
R	0.144	0.396	0.444	0.339	0.119	0.034	0.015
N	0.323	0.190	0.212	0.260	0.220	0.054	0.017
P	0.494	0.198	0.234	0.326	0.642	0.904	0.964

LH-LB	L1	L2	L3	L4	L5	L6	L7
R	0.185	0.373	0.481	0.464	0.304	0.100	0.021
N	0.814	0.626	0.518	0.535	0.695	0.899	0.978

LH-HB	L1	L2	L3	L4	L5	L6	L7
R	0.142	0.432	0.581	0.409	0.068	0.010	0.059
N	0.857	0.567	0.418	0.590	0.931	0.989	0.940

Table 6.6: Average per-Layer Distribution of Node Classes in the First Layers (1000 nodes)

scenarios. We can see in both cases that the position of richer classes is on average always better than poorer classes: in the HH-LB scenario (Figure 6.14a) more than 70% of the VR and R nodes are concentrated in the first five tree layers, while in the HH-HB scenario (Figure 6.14b) the first four layers in average comprise nearly 80% of the VR and R classes. This observation again validates our hypothesis on the relationship between node capacity and the placement in the system, justifying the fast growth of distribution tree fan-out we observed above.

Finally, we collect in Table 6.6 the average proportion between nodes from each bandwidth class in the HH-LB, HH-HB, LH-LB and LH-HB scenarios. We can remark that, in each scenario, the proportion between the node classes found in the first layer (i.e. the peers chosen by the source) approximates quite well the global scenario distribution, as we would have expected, since the source adopts an uniformly random peer selection strategy. Subsequent layers (from L2 to L4-L5) show a definite bias toward the richest classes with respect to the base scenario distribution: in further layers, the likelihood of finding resourceful nodes decreases and becomes quickly negligible.

6.5.2 Asymptotic Behavior of Node Lag

Figure 6.15 represents on a semi-logarithmic scale the average global node lag and its standard deviation, sampled and averaged over thirty seconds after the system reachs steady state. We can see that the lag values for systems with $RI > 1$ are placed on straight lines: this indicates

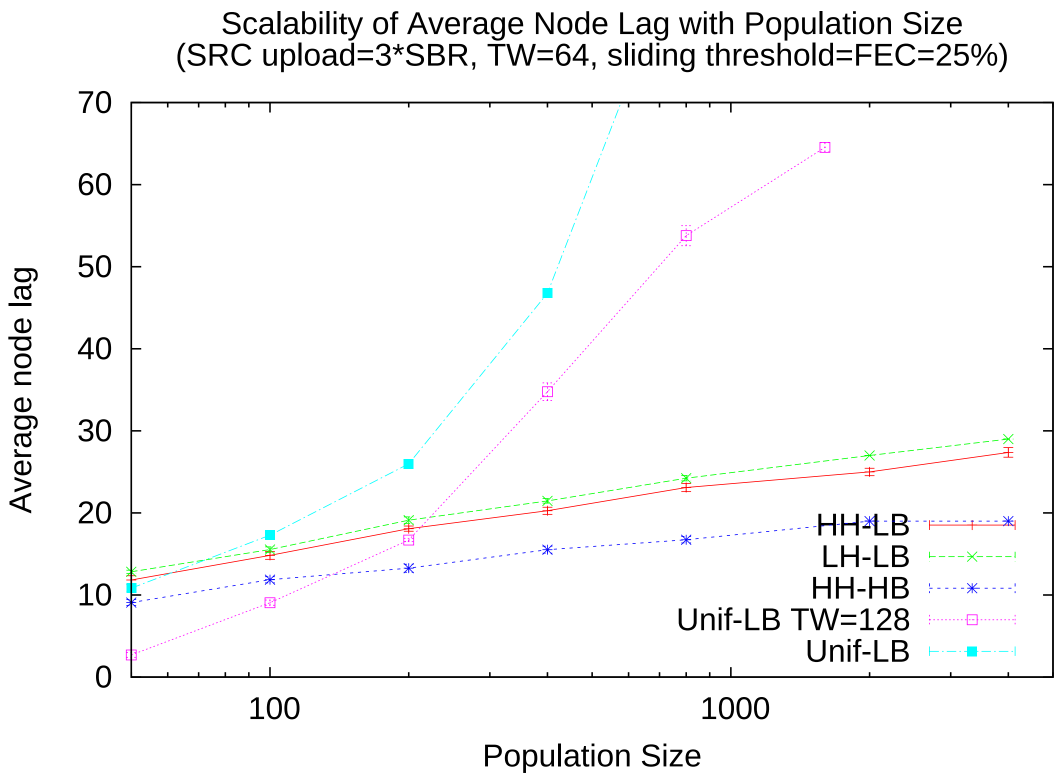


Figure 6.15: Asymptotic Dependence of Average Node Lag on System Size

that PULSE has indeed a logarithmic scalability over the range of system sizes we considered. Interestingly, the logarithm base depends on the bandwidth availability in each scenario, with higher base values - i.e. lower angular coefficient of the lines - for systems where more excess bandwidth is present. The presence of heterogeneity also reduces the average node lag: as we see, the average lag of HH-LB scenarios ($RI = 1.04$) is slightly but consistently lower than the lag of LH-LB scenarios ($RI = 1.2$).

In the same Figure, we provide for comparison the scalability results of uniform systems (Unif-LB scenario) and TW values of 64 and 128. We notice that, despite the small difference in system capacity compared to HH-LB scenarios, systems with uniform bandwidth distribution behave much worse in terms of scalability. The system with $TW = 64$ does not settle on a stable lag for population sizes larger than 400 nodes: while the initial convergence is successful, the average lag slowly but constantly increases. On the other hand, with $TW = 128$, the system manages to settle on stable lag values until 2000 nodes.

The results from the Unif-LB scenarios confirms that larger TW sizes improve efficiency, as we observed above, but also suggests that the condition $\xi \geq 1 - FEC$ alone *is not sufficient* for system convergence. We believe that stability is negatively affected by the incidence of chunk scheduling delays that can add up and result in a discontinuous advancement of node buffer windows. When $RI = 1$, because of the lack of excess resources, the growth rate of these delays as the system grows past a certain scale can no longer be compensated by FEC alone. Non-homogeneous node capacity distributions alleviate this problem: as resourceful nodes are placed near the source and have a very stable lag, they can benefit the whole network by reducing the randomness of the chunk distribution process, making the advancement of other nodes' buffers smoother.

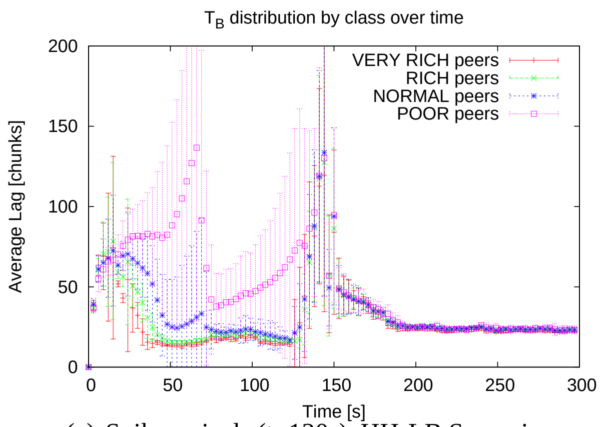
6.6 Results under Dynamic Membership

We have thoroughly experimented with a variety of arrival and departure patterns. We immediately remarked that the impact of exponential churn on the ordinary dynamics of the system was hard to remark, even at high rates. To obtain noticeable effects of node churn, bursts of simultaneous arrivals or departures were required. In Figure 6.16 we show the outcome of four such churn events, again in resource-constrained bandwidth scenarios.

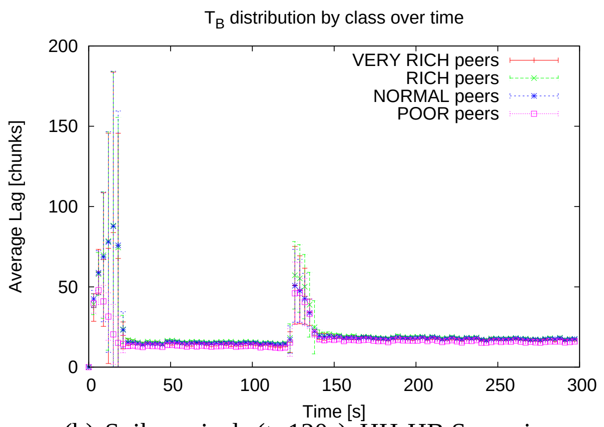
Sudden Node Arrivals In Figure 6.16 we see three sample lag traces of the reaction of the system to node transience. In these plots, nodes can establish up to eight FORWARD connections. We first present the effect of an *instantaneous spike* arrival on our usual bandwidth scenarios. With this arrival pattern, 750 nodes join the network at $t = 120s$, when the initial 250 nodes should have reached steady state. In Fig. 6.16a, we show an HH-LB scenario absorbing a spike of arrivals: it can be noticed that most nodes from all the classes (including the richest ones) are affected and forced to reconnect, but the perturbation lasts for a very short time. By $t = 150s$, in fact, all the classes have reached again convergence. If we increase the available bandwidth, however, the impact of the spike is much reduced. This is the case of Fig. 6.16b, where the HH-HB scenario is shown. Here we can notice that all classes temporarily increase their average lag, but there are no disconnections. The lag increases up to 80 chunks on average but is absorbed very rapidly (in about ten seconds). In this case, the nodes' media play-out is not affected.

Sudden Node Departures Figure 6.16c shows the effect on a HH-LB scenario of the simultaneous departure at $t = 120s$ of 50% of the nodes, chosen at random. We see that the impact of random node departures on the system performance is even lower than from sudden arrivals, as the increase in average lag is minimal and fades away in about ten seconds.

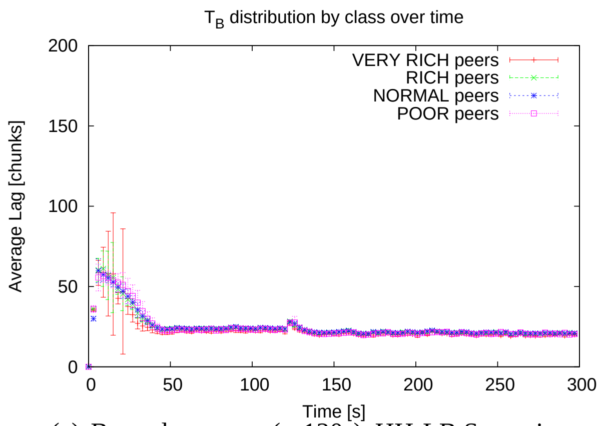
Sudden Resource Shortage To assess the robustness of the system against more extreme forms of churn, we devised a churn scenario where the most resourceful 4% of the HH-LB population, the 40 nodes from the VR class, would suddenly fail at $t = 120s$ and not be replaced. This failure event affects the RI, reducing it to 0.888 (including the source capacity). With $RI < 1$, the system cannot support its current population, whose lag starts to drift linearly. It can be seen, however, that nodes from R and N classes, instead of following the mass of starving P nodes, begin a new convergence phase by $t = 160s$, regaining ground toward the source



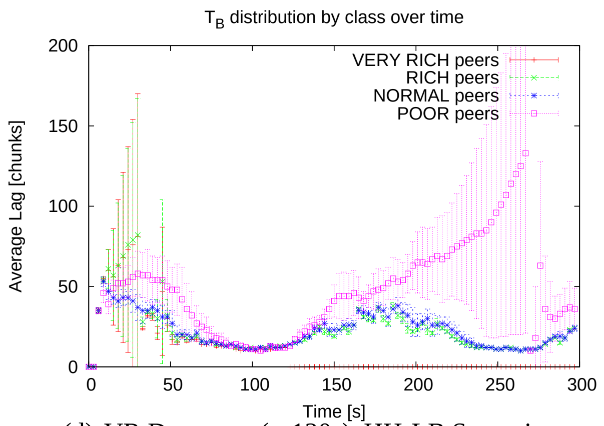
(a) Spike arrivals (t=120s), HH-LB Scenario



(b) Spike arrivals (t=120s), HH-HB Scenario



(c) Burst departures (t=120s), HH-LB Scenario



(d) VR Departure (t=120s), HH-LB Scenario

Figure 6.16: Effects of Node Transience on Global Lag Performances (8 FORWARD)

as they reach average lag values similar to those they had before the departure of the VR class. While the subsequent reconnection by P peers do perturb the recovered system equilibrium, the entire R class and most N peers keep receiving the stream with a reasonable lag. We can also observe in the data trace that a small number of P peers is able to keep its position along the two other classes: however, the majority of P peers that reset their buffers have very few chances of finding again a stable position in the system.

We can appreciate the system evolution under sudden shortage by following the value of Soft Fairness between node classes (Figure 6.17). Before the churn event but after convergence ($80s < t < 120s$), the system operates with a certain degree of unfairness due to the slight excess of capacity ($\mathcal{F}(P,N) \simeq \mathcal{F}(P,R) \simeq \mathcal{F}(P,VR) \leq 0.5$). At $t = 120s$, as the VR class disappears, the Soft Fairness values of P peers versus the other classes get quickly back to the *fair* region ($\mathcal{F} \simeq 0.8$), while the R class keeps only a slight fairness advantage over N: as the difference in node capacity is quite small, these nodes are spread more or less uniformly in the same lag interval. Soft Fairness for the R and N classes noticeably decreases (while still remaining in

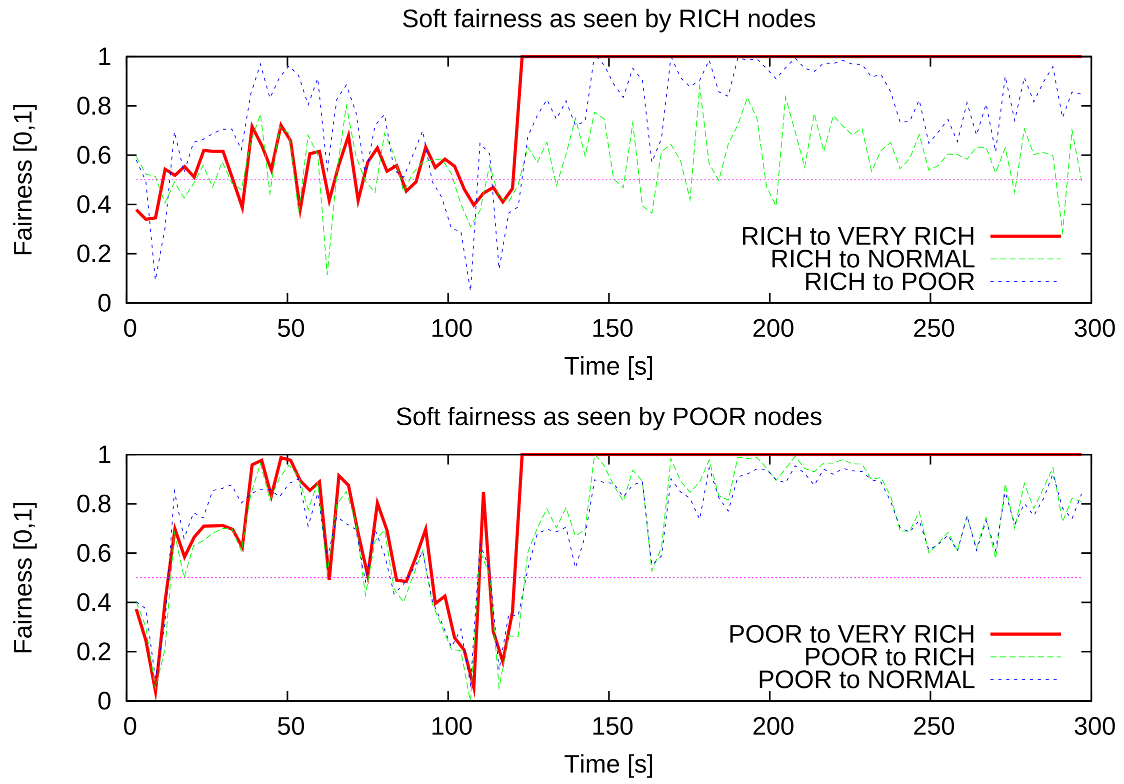


Figure 6.17: Effects on Soft Fairness of Sudden Disappearance of VR Nodes (HH-LB)

the fair region) in correspondence of the reconnection attempt by P peers ($t > 250s$).

6.7 Conclusions

The simulation results presented in this chapter allow us to draw the following conclusions:

Tit-for-Tat as an Optimization Mechanism Our results confirm that the use of a pairwise incentive such as tit-for tat, combined to a performance-based feedback metric such as node lag, provides a powerful mechanism for overlay optimization. Without any explicit knowledge about the capacity of the other nodes, and despite the short average duration of node relationships, each peer reaches and maintains a rather stable lag position inside the system, which in turn results in the timely reception of a steady supply of recent chunks from its neighbors. The analysis of the steady-state data distribution paths indicates that the nodes from resourceful bandwidth classes have the highest chance to be traversed early on during the distribution of each chunk.

Node Lag as a Discrimination Mechanism If we examine a system undergoing resource shortage, we observe that the more resourceful nodes hardly experience any performance degradation. We argue that using the node lag as a discriminating factor for the attribution of altruism helps a running system to withstand the influence of nodes that contribute less than the stream rate. Also, serving neighbors from the MISSING list with higher priority than those from the other lists ensures that altruism will not affect the ability of a node to maintain its current lag even if the altruism is not reciprocated.

The Critical Role of Altruism FORWARD connections, while being established out of altruism, have a clear importance as they reinforce the effect of the tit-for-tat incentive. While the history-based selection process biases the altruism toward those nodes that contributed the most during the past interactions, it does not exclude poorer nodes and free riders. Our simulations show that, by adding few more connections to peers that otherwise would not qualify for tit-for-tat selection, the overall system performance and stability improve dramatically. Richer classes use more of their available upload, overall efficiency increases, and the risk of chunk losses decrease significantly - almost disappearing with eight or more FORWARD connections.

Altruism as a Rational Choice (for both, Own and Common Good) When FORWARD connections are disabled or reduced, we notice that especially the richest peers are often contributing less than what they could actually offer to the system. The partial use of their capacity, while allowing them to maintain regular relationships with other resourceful peers, exposes them to the risk of starvation, caused for instance by transient content bottlenecks. Starvation happens for two reasons: a direct issue of competition between resourceful nodes, which makes the outcome of future peer selections unpredictable, and an indirect problem of availability of new chunks, which are spread to only a small number of nodes chosen using the tit-for-tat criterion. By spreading additional copies of recently generated chunks, the richer nodes can solve these two problems at the same time: competition is eased, as the number of peers that are eligible to select the richer nodes becomes larger, and starvation becomes less likely, as new chunks are made available much faster.

Chapter 7

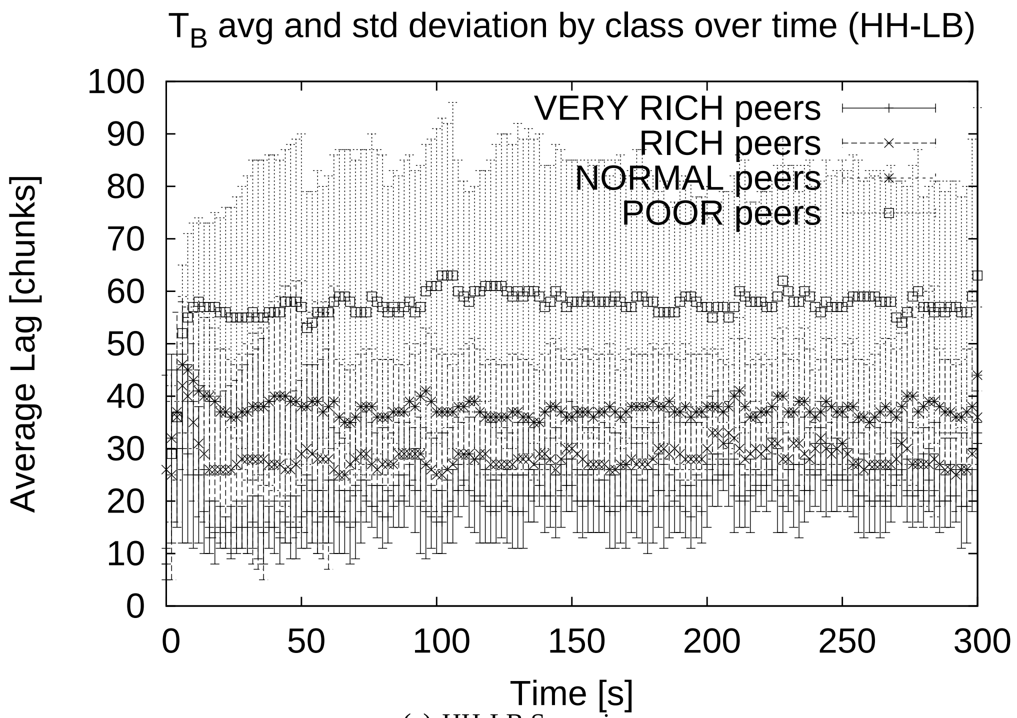
Experiments and Real Measurements

This chapter[1] completes the evaluation of PULSE with additional results, obtained from the emulation of PULSE systems over large-scale network testbeds.

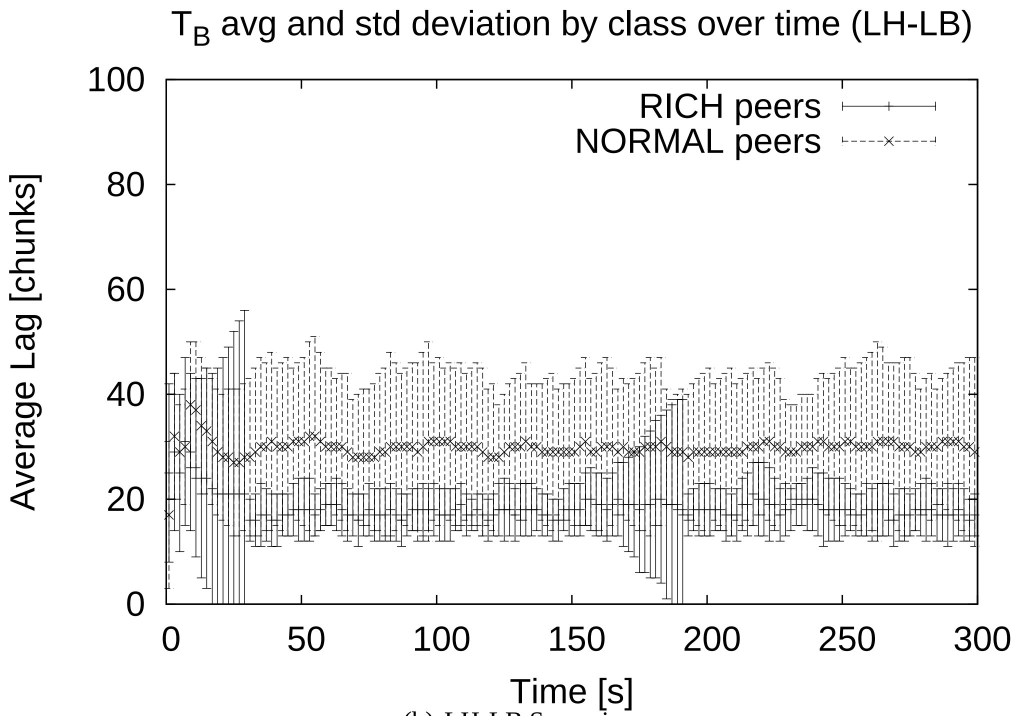
7.1 Validating the Simulation Results

We used the Grid'5000 [3] large-scale testbed to perform a qualitative validation of our simulation results. Grid'5000 currently offers 700-800 hosts (about 1200 CPUs) physically located in several French research institutions and connected by a fast backbone network. We tried to approximate on this testbed the model used in PulSIM, i.e. negligible and constant latency between nodes, no bottlenecks in the core network, and controlled attribution of access capacity to individual nodes. As the pairwise latency between the testbed nodes was very low (in the order of several milliseconds) and uniform, we performed our experiments using the bare underlying network topology of the testbed. We implemented configurable upload bandwidth caps into the node prototype software, in order to emulate the effects of upload bottlenecks at the access links: we use a token-bucket algorithm where a variable number of tokens (depending on the bandwidth class of the node) are replenished every 125 ms.

We managed to reach a maximum scale for our experiments of 1000 simultaneous nodes by running up to two peers on each testbed machine. The protocol parameters used in the Grid'5000 testbed evaluation are listed in Table 7.1: having reduced the chunk rate R to 8 chunks per second, that is half rate compared to the simulations, the length in time of the node TW is now doubled, that is 8 seconds of media. Because of technical limitations, to emulate the initial simultaneous node arrival we sequentially launched the nodes in parallel batches of 200. The time required to launch 1000 nodes with this technique was usually less than 20 seconds.



(a) HH-LB Scenario



(b) LH-LB Scenario

Figure 7.1: Testbed Validation of the PULSE Prototype Node: Class Lag over Time

Parameter	Value	Description
W	32	Length of buffer sliding window [*chunks*]
TW	64	Total length of trading window [*chunks*]
LR_{max}	20%	FEC tolerance to chunk losses/window
T_D	$\frac{250}{R}$	Minimum lag to trigger buffer reset [s]
EPOCH	2	Time b/w subsequent peer selections [s]
N_{TFT}	4	Peers chosen as MISSING neighbors
N_{FWD}	8	Peers chosen as FORWARD neighbors
R	8	Rate of chunk generation @source [s^{-1}]
SBR	256	FEC-encoded stream bit rate [$Kbit/s$]
$\mathcal{R}_{TO}$	0.5	Timeout of chunk request messages [s]
$\mathcal{R}_{max}$	2	Max outstanding requests to same peer

Table 7.1: PULSE Protocol Parameters Used for Testbed Experiments

7.1.1 Convergence and Evolution of Node Lag

We launched extensive simulations of PULSE systems, mainly concentrating on the HH-LB and LH-HB scenarios. In Figure 7.1 we show two sample traces from two 800-node deployments under the HH-LB (a) and LH-LB (b) scenarios. As we expected, the impact of the initial transitory on the emulation results is much lower than what we observed in the simulated outcomes: in fact, no massive "reset and reconnection" event was ever detected during the earliest phases of every deployment. If we check in the peer logs the amount of peers that reset their buffer in the HH-LB scenario, we notice that only a handful of resourceful peers get disconnected early on during initialization (2 RICH and 2 NORMAL), compared to several POOR peers (about 40 in the first 50 seconds). Few POOR nodes happen to regularly disconnect during steady state, for a total of 100 reset events during the 300 seconds shown above.

Figure 7.1 confirms the strong relationship we had observed in the simulations (Sections 6.4 and 6.5) between the available upload bandwidth of a class and the average lag of its members. The peers with the highest bandwidth contribution reach in both scenarios a steady-state lag of about 20 chunks (that is, less than 3 sec) 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 (slightly more than 7 sec). 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 difference in node lag between classes becomes higher when the available upload capacity of a class is smaller than the stream bandwidth*. The same remarks can be made regarding the LH-LB scenario, as the difference in average lag of the two classes is smaller but still evident (18 chunks or 2.2 sec for RICH, vs. 30 chunks or less than 4 sec for NORMAL).

Figure 7.1 also shows the standard deviation of node lag as vertical lines. Here we can notice

[1]The contents of this chapter have been published in part as [86].

more clearly than in the simulations how nodes have their lag distributed around the average class lag: the variance of T_B appears as strongly dependent on node contribution and becomes higher as the upload bandwidth available to each class decreases. This phenomenon can be observed both in the HH-LB and in the LH-LB scenarios: the standard deviation of the lag is 4-10 chunks for VERY RICH and RICH classes, vs. 10-25 chunks for NORMAL and POOR in HH-LB; 4 chunks for RICH vs. 11 chunks for NORMAL in LH-LB. This fact suggests that *having more bandwidth not only reduces the average lag, but also tends to give nodes a more stable performance in the system*.

We can in fact appreciate how the average T_B for each class is very stable over time in both traces, with minimal fluctuations. The biggest event that can be observed (between $t = 160s$ and $t = 190s$ in the LH-LB plot, a point where the standard deviation of the RICH class increases and then falls back to the previous values), is caused by a temporary increase in the lag of eight RICH nodes, with two of them reaching a peak T_B of 80 chunks before recovering their previous lag value. In this scenario, no node suffered data loss.

7.1.2 Bandwidth Classes and Data Paths

After validating the main qualitative properties of PULSE behavior in a realistic context, we are now interested in observing the distribution process of individual data chunks and in comparing these results to our expectations. To this end, we will again study the paths taken by data chunks as they are replicated by the nodes

Figure 7.2 contains the analysis of the average properties of chunk distribution trees from our traces shown above. We notice that the maximum tree depth in hops for individual chunks is short and quite stable over time. In our system with 800 nodes, maximum tree depths are in average between 11 and 14 hops, for both bandwidth scenarios. We can also notice how HH-LB trees are on average as deep as LH-LB trees and equally wide up to the 6^{th} layer, despite the fact that the Resource Index for the HH-LB scenario is much lower than for LH-LB.

Comparing Figure 7.2 with Figure 6.13 in the previous chapter, we can validate the fact that the first few layers of the trees are in average very wide, even if not as wide as in the simulated systems, and that the relative system behavior for the two scenarios is quite similar in both cases. We also confirm that the paths taken by the chunks are consistently good, even under widespread bandwidth scarcity and while the data connections between nodes are continuously renegotiated.

The tree depth values observed in the emulation are slightly higher than predicted by our simulations, but still in the same order of magnitude (for 800 nodes, the average maximum depth of HH-LB trees would have been in average about 9 hops and less than 7 hops for LH-LB). The difference with respect to simulation is likely due to the reduced efficience of chunk exchanges in an asynchronous environment with additional delays for control message exchange. Finally, the rare chunk losses, revealed by trees significantly shorter than the average, appear to be more frequent here than in the simulated cases, but do not result in playback disruption.

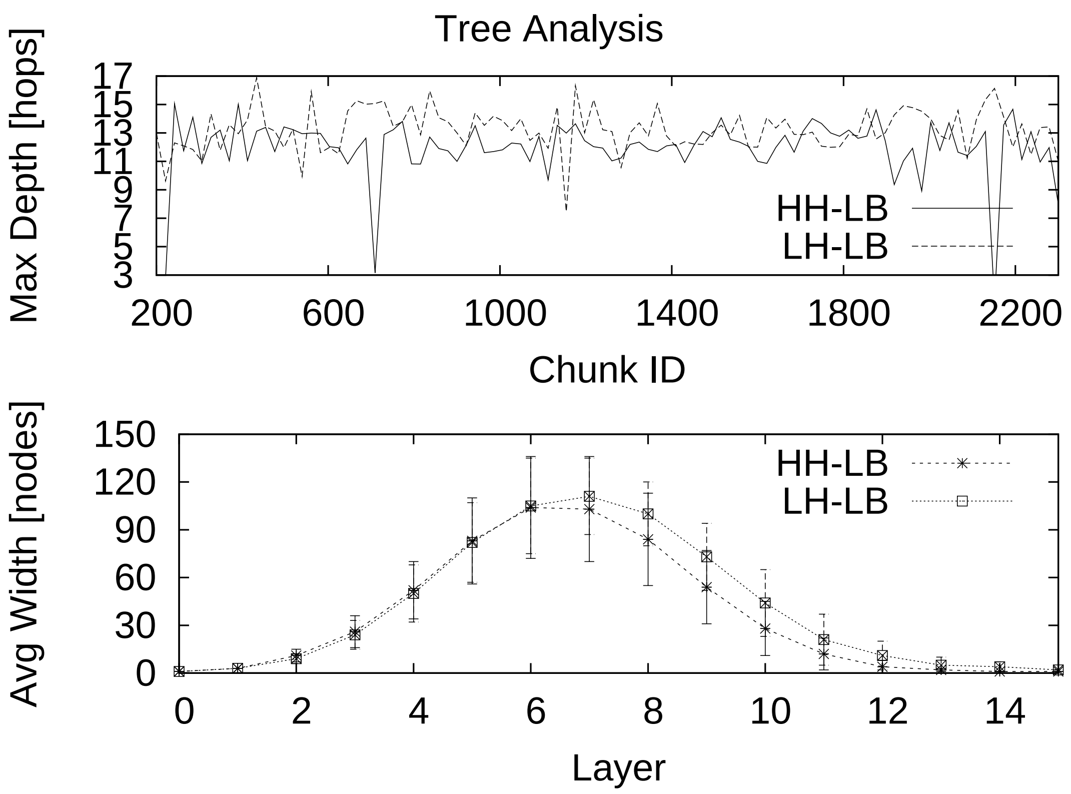


Figure 7.2: Analysis of Average Chunk Distribution Tree Properties (800 nodes)

7.1.3 Interactions between Bandwidth Classes

We now examine the consequences of peer selection in terms of the amounts of data exchanged. We keep track, for each chunk, of the class of its sender and receiver peers, and of the type of the connection, i.e. whether it's MISSING, FORWARD, or NEW, as determined by the sender. In Figure 7.3 we display the data we collected over a typical HH-LB run, as seen from the point of view of both the uploads and the downloads.

The first finding is that, for every class except POOR peers, MISSING exchanges in upload do convey on average more or less 80% of the stream bandwidth. Even when nodes may be able to provide much more, their capacity is not used for MISSING exchanges, and would probably be wasted to a large extent if no FORWARD connections were present. We also notice a striking similarity between the amounts of data uploaded and downloaded by each class using MISSING exchanges. This observation suggests that increasing the number of MISSING connections could only marginally increase the amount of data they can convey, as the stream rate is a natural barrier for reciprocal contribution.

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 VR and R peers alone to nodes in the POOR class provide, in average, almost half of their stream rate, while they obtain the rest from MISSING exchanges, largely with other POOR peers. On the other hand, the

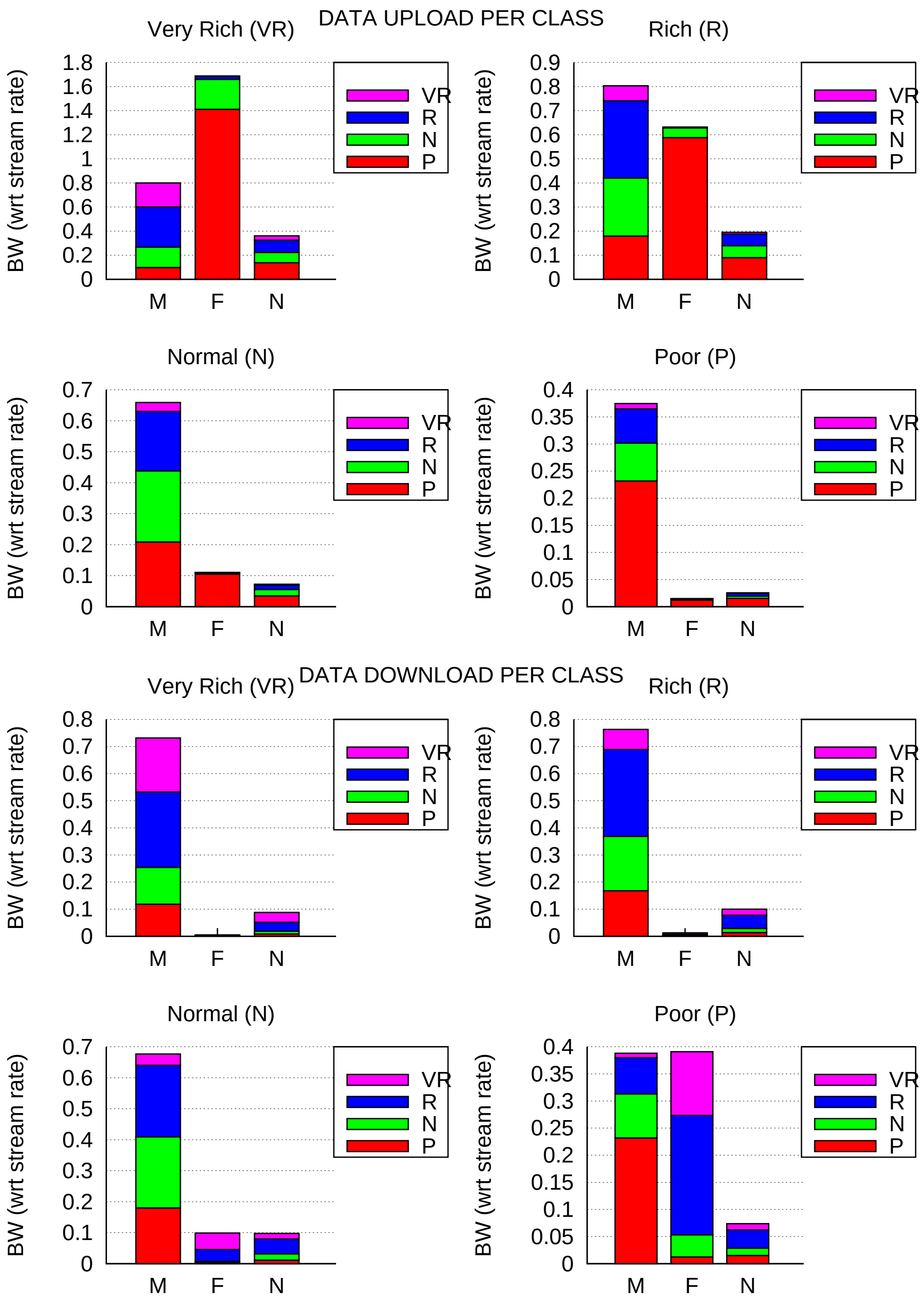


Figure 7.3: Average Cumulative Data Exchange Outcomes by Bandwidth Class (G5K, HH-LB)
*Amounts of data exchanged are broken down by connection type (M=*MISSING*, F=*FORWARD*, and N=*NEW*) and normalized by the stream rate.*

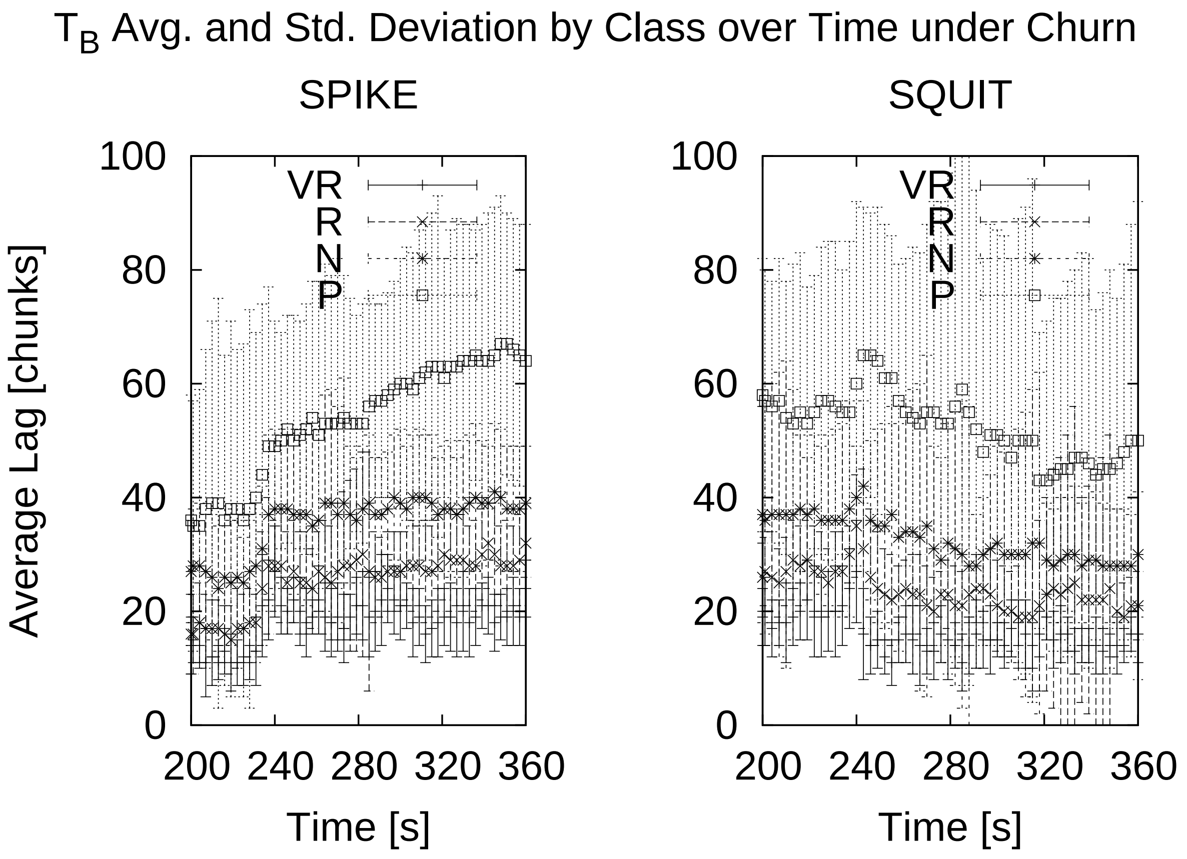


Figure 7.4: Average Class Lag over Time for HH-LB under SPIKE and SQUIT

scarce bandwidth of the two poorest classes is rarely allocated to FORWARD or NEW connections.

These results validate the fact 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 node lag and History score allow to distribute the unused resources evenly to the entire system.

7.1.4 PULSE under Churn

We have reproduced the churn scenarios we used in our simulation in the context of our testbed experiments. In Figure 7.4 we show the effects of the SPIKE and SQUIT churn scenarios on the average lag of each bandwidth class of an HH-LB system. The churn event is scheduled at $t = 230s$. We can observe how the system handles gracefully both events, with a visible increase (decrease) of T_B over the seconds immediately following the arrival (departure) that quickly leads to new stable average lag values. We remark that the response of real systems to massive arrivals is very fast, while departures require some time for the system to reach a new stable configuration: the time constant of the SQUIT scenario likely depends on the presence of excess capacity in the system, which nodes can exploit in order to reduce their average lag. Visually, the effect of both churn events is completely absorbed after about 150 seconds from their occurrence.

7.1.5 Results of PlanetLab Deployment

Next, we performed experiments on PlanetLab in order to observe the behavior of PULSE under uncontrolled network conditions. In fact, PlanetLab offers its users little control on the node resources, not only in terms of an unknown available bandwidth, but also because of the high CPU load of machines. This problem makes it especially difficult to test a time-sensitive streaming application that requires low response times. For this reason, we only managed to find about 200 hosts with semi-acceptable CPU load conditions, while we had to lower the chunk rate R to 4 chunks per second to reduce the amount of processing performed by the nodes. We conducted experiments with bitrate values ranging from 64Kbps to 512 Kbps, and performance appeared to be rather insensitive to the stream rate up to 256 Kbps, decreasing for further bitrate values. This phenomenon may be due to traffic shaping or other counter-measures adopted by PlanetLab nodes to limit a steady bandwidth consumption by individual users. As we could not collect enough data for an in-depth analysis of this issue, we rather decided to avoid it altogether by using a “safe” low bitrate: therefore, we set the stream bitrate in our experiment to 128 Kbps.

Furthermore, we resolved to not artificially constrain node bandwidth, but rather chose to leave it naturally limited by the resources available at each host (as the high CPU load in most PlanetLab nodes would often interfere with the execution of the software in unpredictable ways). As bandwidth classes were not defined for this experiments, we instead collect information such as the cumulative amount of data uploaded, which will be useful to reconstruct the relationship between node lag and contribution. As we see in the lower plot of Figure 7.5, no node contributed alone an extraordinary amount of resources to the system: rather, 99% of node contributions (averaged on the whole system lifetime) ranged between zero and 2*SBR. We argue that these values can be considered a reasonable approximation for an Internet-based streaming event.

Looking at Figure 7.5, 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 can obtain in Figure 7.5 a clear inverse relationship between the two variables: the more a peer uploads, the lower is its lag. The results we obtained from this experiment show that PULSE behaves reasonably well even a difficult environment such as PlanetLab, proving again a high level of resilience and adaptiveness.

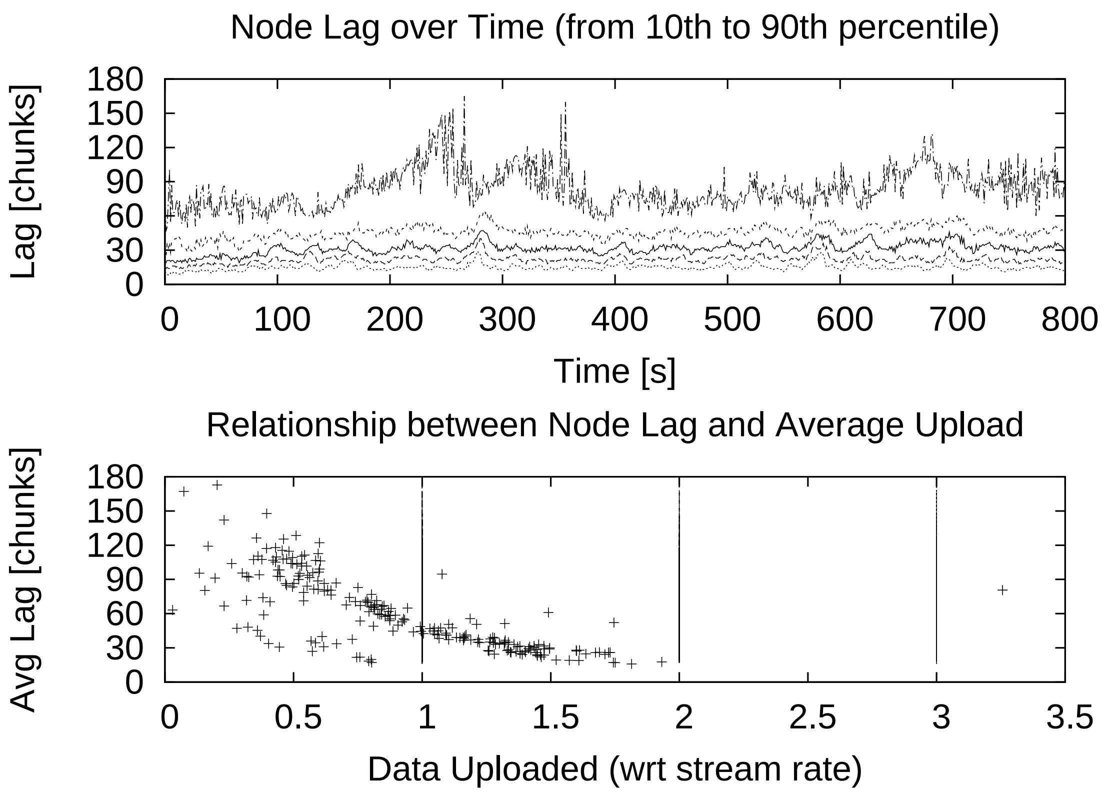


Figure 7.5: Results of an Uncontrolled PULSE Run on PlanetLab (200 nodes)

7.2 Evaluating Latency Awareness

After examining the macroscopic effect of tit-for-tat peer selection on the global system organization, we now study the impact of a weighted latency bias on the system in terms of awareness to network locality. These experiments were performed on Planetlab using a population of 100 peers, again without any artificial upload limitation. We observed the behavior of the PULSE system as the C latency weight parameter was set to 0 and 1. We used the pairwise node latencies measured during our experiments with exponentially-spaced *ping* probes ($\lambda = 10s$) as a practical network locality metric.

Average Latency of MISSING Connections Figure 7.6 shows the *average total latency* of the incentive-driven connections in function of the average lag of each peer. Average total 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 connection delay, especially for those peers whose lag is low. We can see that, when $C = 0$ (i.e with no latency bias), all peers show an average cumulative latency uniformly distributed between 45 ms and 60 ms, regardless of their average lag. With the addition of a latency bias $C = 1$,

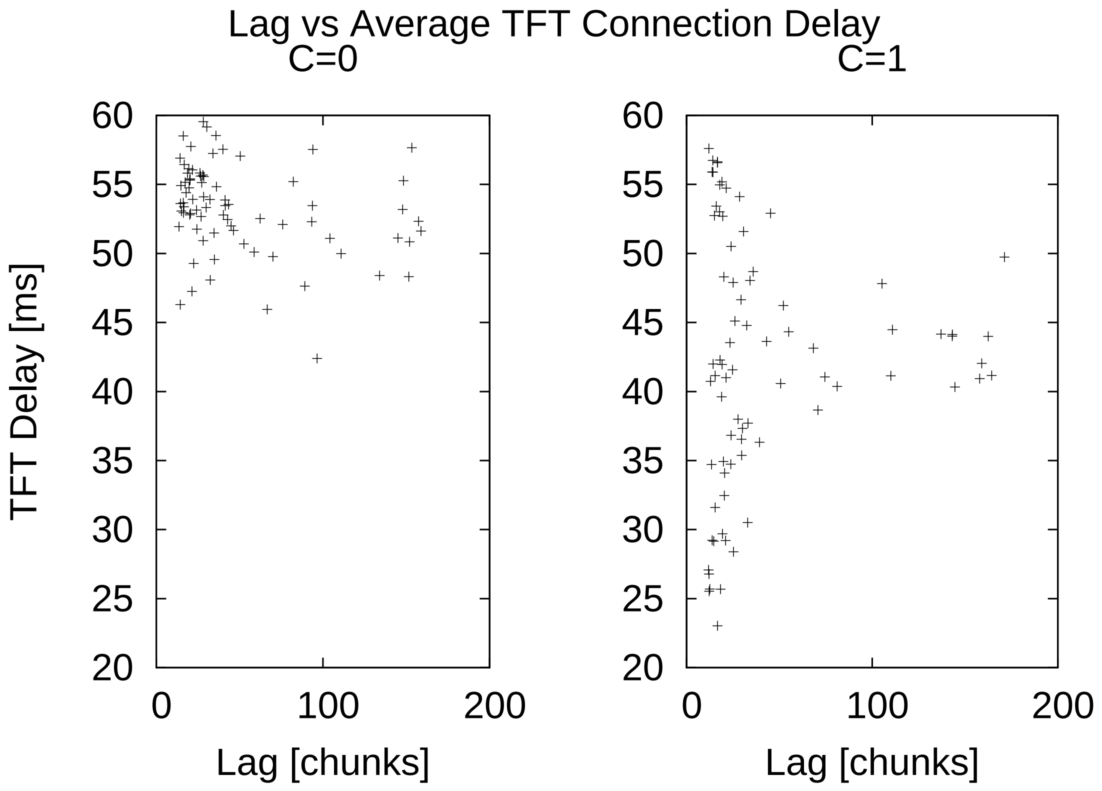


Figure 7.6: Effect of Latency Bias on Cumulative Connection Latency

the minimum average cumulative latency goes down to 22 ms, while just few nodes maintain a cumulative latency of about 60 ms. Also, the average cumulative latency of all nodes becomes lower for non-zero values of C.

Estimating the Locality of Data Exchanges In Figure 7.7 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 $C = 0$, 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 $C = 1$, 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.

Other Advantages of Locality Optimization Finally, Table 7.2 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. We expect that, by introducing the latency bias in a scenario with larger difference

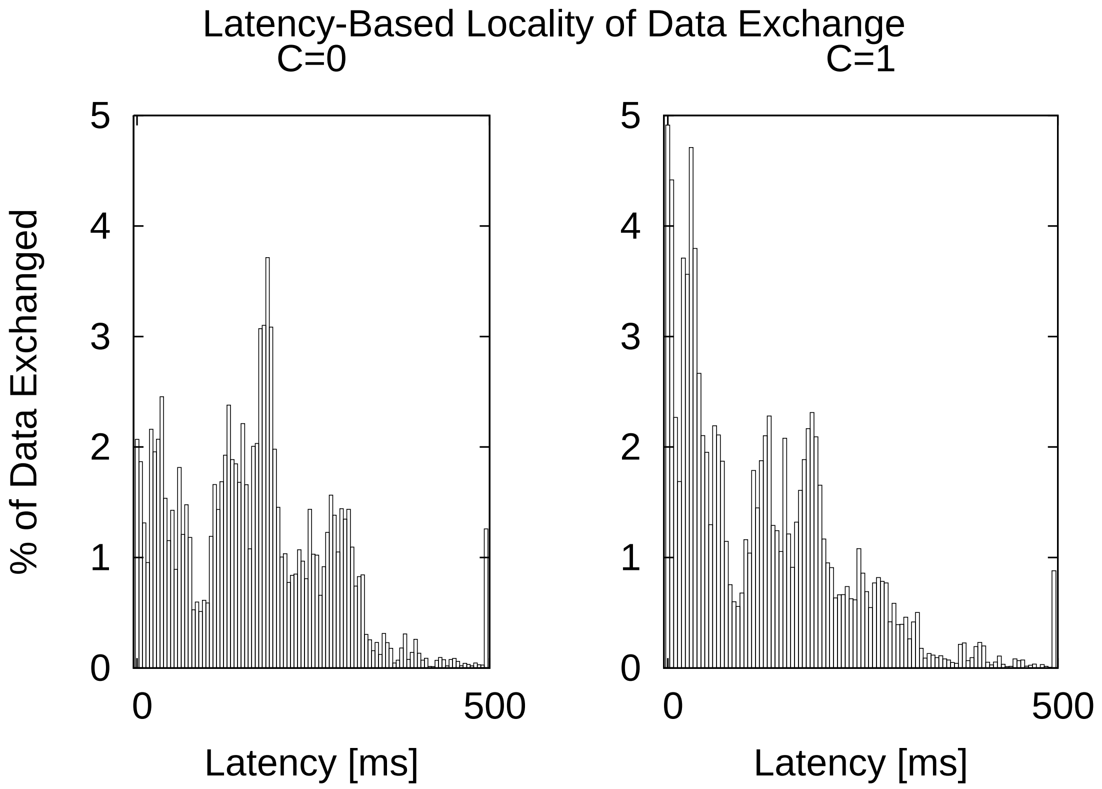


Figure 7.7: Effect of Latency Bias on Overall Data Exchange Locality

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

Table 7.2: Effect of Latency Bias on Average Node Lag (in chunks)

between pairwise latencies, the reception delay reduction would also be bigger.

7.3 Conclusions

In this chapter, we validated our analysis of the behavior of PULSE using emulation: we ran a number of controlled experiments on a large-scale grid testbed, reaching a population size of 1000 nodes, and reproduced the same scenarios we previously used for our simulations in order to make a direct comparison. We could observe a substantial similarity between simulation results and the actual behavior of emulated systems: the presence of network delays and asynchrony in the emulated environment greatly improves the visibility of aspects such as stability of node class lag and system-wide response to churn. We then described our results from PlanetLab experiments, which are quite encouraging as they confirm the subsistence of a definite relationship between node lag and upload contribution even in uncontrolled environments, where both the RI and the node capacity distribution are not known a priori.

Finally, we evaluated the effectiveness of introducing pairwise latency as an additional criterion for peer selection: the use of a simple *latency bias* in the optimistic choice of MISSING neighbors led to noticeable improvements in the overall locality awareness of MISSING connections and total data exchanges. Furthermore, we could measured in our experiments a net decrease in the average node lag of at least 20% when latency awareness was enabled: this result suggests that building some form of locality awareness into data-driven live streaming systems can provide great practical advantages in terms of both, efficient network usage and performance gain.

Chapter 8

Conclusion

Internet-based peer-to-peer live streaming systems have to face several technical challenges. They must be capable to scale to large populations, support highly asymmetric distributions of node capacity, and withstand arbitrary user behavior. This thesis presented and evaluated PULSE, a practical unstructured data-driven P2P system for live media streaming. The PULSE algorithms exploit the relative flexibility in the timeliness requirements of live streaming in order to improve the performance and the robustness of the system in real-world conditions. The results we obtained from simulation, testbed emulation, and Planetlab experiences confirm that PULSE can meet the challenges of a full-scale Internet deployment.

8.1 Contributions

PULSE makes several contributions: its design combines an unstructured mesh-based architecture, which grants the nodes great freedom to associate, with pairwise incentive mechanisms, which are used as peer selection strategies. Thanks to the use of incentives to optimize bandwidth allocation, combined to dynamic peer selection strategies that rely on implicit feedback from data reception, PULSE is capable to support a operate in non-cooperative environments. The presence of a global control loop based on *node lag* allows PULSE to quickly react to changes in both, node resource availability and system membership.

While the incentives in PULSE do not aim to *directly enforce fairness*, as the common-sense interpretation of the word “incentive” may suggest, they are meant to redistribute node capacity in an useful way and, in case of system-wide resource shortage, to favor the nodes that contribute more to the system in order to preserve the health of the global system. We also argue that the risk of buffer starvation during live streaming, which becomes high for peers that offer an insufficient upload contribution, is an *indirect incentive* to cooperation that should be appealing to rational nodes/players.

The simulation results we presented show that an altruistic incentive-based peer selection fosters cooperation among the nodes and leads to the emergence of clusters of peers with similar resources. The presence of resource-aware clustering is functional to the efficient and timely distribution of data throughout the system: we observed that in PULSE the number of hops that a chunk takes, on average, is comparable to the height of trees in tree-based architectures, while the first layers of the chunk distribution trees can be significantly wider.

The experiences we performed using large-scale testbed emulation and PlanetLab deployments, in addition to validating the simulations, indicate that PULSE is capable to operate in a harsh environment such as the Internet. Also, they show that PULSE can easily take into account network locality when establishing node relationships, introducing a further performance improvement. Our results also confirm the ability of unstructured mesh-based systems to withstand the high levels of transience that can result from user and network dynamics (churn, failures, congestion, etc.).

Finally, we have collected and elaborated a number of metrics to evaluate both generic and resource-aware data-driven systems. Besides their immediate interest for the analysis of PULSE, we hope they will help to reach an improved understanding of unstructured streaming systems and to develop a more comprehensive comparison between structured and unstructured approaches to live data distribution.

8.2 Outlook

The lessons learned from the design, simulation, implementation, testing and initial deployment of the PULSE system suggest that *ubiquitous P2P live streaming on the Internet is indeed possible – and does not require a large amount of resources at the involved Internet nodes*. The first practical streaming architectures have shown the feasibility of the end-system multicast concept. The current generation of large-scale P2P systems indicates that audiences of several hundred thousand users can be served by an adequate provisioning at the source (upload capacity from tens of Mbps[1] up to several hundreds of Mbps[2]) and using advanced peer selection and chunk selection algorithms. The next wave of live streaming applications could realistically involve a multitude of broadcasters that serve media to populations that can range in size from few to tens of thousands of users, just leveraging standard computing equipment and broadband Internet access.

The Road Toward Optimality The search for an optimal strategy to distribute live data over networks with arbitrary upload capacity distribution is not over yet: while PULSE is among the first to explore pairwise incentives as a non-deterministic optimization technique, the recent developments of theoretical models [72][34] and the application of graph theory [45], game theory [75] and gossip algorithms [95] will certainly lead to an improved understanding of the live streaming problem and to the design of new, more efficient algorithms and solutions. It is our opinion that the improvement of live streaming techniques will also benefit applications for

[1]Gale Huang, PPLive Software Architect, Keynote at the 2nd SIGCOMM P2P-TV Workshop, 2007
[2]Qian Zhang, Hong Kong University of Science and Technology, *ibidem*.

large-scale data diffusion, such as video on demand and bulk file distribution. The deliberate introduction of loose synchronization between receivers, once seen as an undesirable constraint of live media, could actually prove an effective method to increase the efficiency and data distribution performance of these systems.

More Focus on Non-Cooperative Scenarios The live streaming problem has mostly been studied in cooperative contexts so far, which implied either *full compliance by the nodes to system policies* (e.g. providing as much upload as required, connecting to a specific number of “entitled” sub-trees), or at least *honest reporting of protocol information* (e.g. correctly describing the number of children served, providing a truthful account of the content a node buffer). PULSE is one of the first practical systems for live streaming to mitigate the consequences of non-cooperative behavior (as freeloading or insufficient upload contribution): the support for bandwidth heterogeneity naturally implies the ability to react to purposeful lack of cooperation.

Nevertheless, many avenues of attack emerge when the hypothesis of policy-compliant behavior is rejected: an interesting example is provided by the latest attempts to defeat other incentive-based mechanisms, such as the BitTorrent protocol [68][88]. While the tit-for-tat strategy has not been surpassed yet by any other strategy and thus appears to be Pareto-optimal, its implementation in a distributed protocol for data exchange opens up several weaknesses [100]. In the case of deliberate tampering with protocol information, however, the problem becomes way more complex, reaching to the realms of practical and theoretical computer security: furthermore, the assumption of player rationality may no longer hold if nodes try to actively disrupt the system without seeking any benefit from it. The concept of *faithfulness* [101] of an application as a form of resilience to misleading external information and behavior has been recently introduced to support the design and analysis of P2P systems.

The Future of PULSE The software for the PULSE node has been released to the public by France Telecom R&D under a LGPL Free software license in early 2007. The current codebase for our node prototype (v. 0.1.1) is substantially the same that we used to obtain the traces for our latest published papers [86][87] with some added bug-fixes. The future development of the public branch of the PULSE node software shall be continuing as a part of the European project (STREP) NAPA-WiNe. Moreover, the use of a Free license allows and encourages the contribution by individual developers around the world.

Chapter 9

Synthèse en Français

9.1 Introduction

Cette thèse est dédiée à l'étude du problème de la diffusion de flux vidéo en direct sur Internet, mieux connu sous le nom anglais de "live media streaming". Nous avons choisi de concentrer nos efforts sur ce problème pour plusieurs raisons.

Première raison, la distribution de données sur Internet vers de grands nombres de récepteurs est aujourd'hui un besoin fondamental: les efforts dédiés pendant les dix dernières années à la diffusion de contenu statique (i.e. fichiers) ont laissé la place à une attention accrue pour la distribution de flux de données dynamiques. L'émergence des systèmes pair-à-pair (en anglais *peer-to-peer*, abrégé en *P2P*) dans le contexte de la distribution de données a permis une importante amélioration des performances par rapport aux architectures basées sur serveurs centralisés, surtout en termes de réponse à la croissance de la taille du système (aussi défini *passage à l'échelle*, provenant du mot anglais *scalability*). Le principe fondamental des architectures P2P est l'équivalence de rôle (*a priori*) entre toutes les entités qui composent le système, qui sont appelées pairs ou nœuds.

Deuxième raison, l'extension d'une approche P2P au streaming live est une étape conséquente de l'évolution des techniques pour la distribution de données, qui ajoute de nouveaux défis au problème initial, notamment le respect de deux contraintes: l'ordre de réception et le délai de reproduction (en anglais, *playout delay*) du flux vidéo. Troisième, le problème du streaming live est clairement défini: puisque les nouvelles données sont générées constamment par la source *(i)* tous les recepteurs sont à peu près synchronisés dans la reception du flux, *(ii)* le système a une faible mémoire des événements passés et *(iii)* ses performances peuvent se révéler moins dépendantes du comportement des utilisateurs par rapport à la distribution de fichiers ou à la distribution de vidéo à la demande (*VoD*). Quatrième, l'interêt commercial pour des solutions de streaming live qui puissent passer à l'échelle avec de faibles coûts est à présent très fort, puisque les conditions nécessaires pour le succès et la diffusion de ces applications sont désormais satisfaites (ordinateurs personnels avec puissance de calcul suffisante, appareils photographiques

Type	Application d'Exemple	Contraintes de Distribution	Contraintes de Reproduction
Bulk	BitTorrent, E-donkey	Aucune (fichiers vidéo)	Video reproduite après réception
VoD	Joost, Youtube	Aucune (fichiers vidéo)	Reproduire pendant la réception (1 min)
Live	Peercast, PPLive	Court délai (10 $\sim$ 30 s)	Reproduire au plus tôt (10s)
Interactive	Conference, Skype	Immédiate ($\sim$ 100 ms)	Reproduire tout de suite (100 ms)

Table 9.1: Applications pour la Diffusion de la Vidéo

et *webcams* peux coûteux, facilité d'accès à l'Internet avec hauts débits). Enfin, nous sommes convaincus de la présence de plusieurs aspects qui peuvent être améliorés dans les systèmes P2P existants, et du fait qu'il nous est possible de contribuer à leur amélioration.

9.1.1 Définition du problème

La Table 9.1 présente une vue globale sur l'ensemble des applications pour la diffusion des données multimédias. Tandis que la vidéo à la demande (VoD) consiste en la distribution de données qui ont été enregistrées (par exemple des fichiers vidéo) de façon qu'il soit possible les reproduire pendant qu'ils sont récupérés, le streaming live est la première application pour qui les données constituent un flux qui n'a pas de début ni de fin, mais qui continue d'evoluer au fil du temps. Les flux video en direct sont differents des flux VoD car:

- Les données d'un flux en direct sont distribuées par la source vidéo uniquement pendant un e courte période, après laquelle elles cesseront d'être disponibles
- La durée totale d'un flux en direct n'est pas connue a priori: typiquement, les utilisateurs vont rejoindre le système quand une session de streaming est déjà en cours et vont la quitter avant qu'elle soit terminée. Le temps passé par un utilisateur dans le système peut être considérée comme négligeable par rapport à la durée totale du flux (d'où la définition de durée pratiquement infinie d'un flux de streaming live)
- Les destinataires d'un flux sont intéressés à le reproduire avec un délai raisonnable, puisque l'intérêt pour ses données courantes est très volatil.

Le streaming live est une application qui pose plusieurs difficultés dues à la présence de contraintes de délai. Au même temps, ces contraintes laissent un certain marge de liberté au concepteur de systèmes par rapport au cas des applications interactives, ce qui rend le problème intéressant par la variété des choix possibles au niveau de l'architecture: les quelques dizaines de secondes qui découlent entre la génération des données a la source et la reproduction aux récepteurs peuvent être suffisantes pour que les données soient traitées entre temps par plusieurs pairs et redistribuées avec une série de sauts (*hops*) sur le réseau.

Grâce à cette marge de tolérance au retard, le streaming live permet l'adoption de nombreuses solutions techniques, notamment les architectures P2P. Une approche P2P a l'avantage de permettre au système de passer facilement à l'échelle, puisque chaque nœud du réseau P2P, au même temps qu'il consomme les ressources du système, peut aussi mettre ses propres ressources au service d'autres nœuds. En principe, si chaque participant contribuait au moins autant de ressources qu'il consomme, le système P2P pourrait atteindre des tailles arbitraires. Cette propriété, appelée en anglais "self-scalability", est la raison fondamentale qui justifie l'application d'une approche P2P au problème du streaming live. De plus, dans une architecture client-serveur, le coût du streaming est totalement au frais du fournisseur du contenu, puisque cela demande l'allocation d'une quantité de ressources du coté du serveur qui est *proportionnelle à l'entité maximale prévue* de la population à desservir. Au contraire, dans une architecture P2P, la source du contenu ne requiert qu'une faible quantité de ressources, qui est largement indépendante de la taille du système. Les avantages économiques de cette propriété sont aussi évidents, surtout quand les populations a desservir sont très larges. Au cours des dernières années, plusieurs architectures P2P pour le streaming live ont été proposées. Le Chapitre 2 de cette thèse étant dedié à l'analyse et à la comparaison entre les architectures principales des systèmes dans la littérature du secteur, nous nous limitons ici à énumerer les défis techniques auquels les systèmes P2P pour le streaming live doivent se confronter.

Passage a l'échelle Bien que les ressources augmentent avec le nombre de nœuds dans le système, une limite supérieure théorique de taille existe pour les applications sensibles au temps, tels que le streaming live. Cette limite est due au retard introduit par chaque "hop" qui est traversé par les données du flux. En conséquence, le principal défi du streaming P2P est de mettre à point une technique de distribution de données qui garantit une distribution continue et permet d'éviter une excessive accumulation de retard quand le nombre d'utilisateurs augmente.

Adaptation aux ressources Un autre défi par rapport au passage à l'échelle dérive de l'hypothèse que chaque nœud contribue suffisamment de ressources au système. En realité, les nœuds peuvent contribuer moins que ce qu'ils consomment, voire rien du tout, soit en raison de limitations techniques inhérentes (scarcité de ressources, présence d'un pare-feu, etc.), soit par malveillance, ayant fait le choix explicite de tricher (*freeloading*).

Équité dans le service Si l'hypothèse de la coopération spontanée n'est pas maintenue, de nombreux problèmes (tels que le rejet de nouveaux utilisateurs ou la perte de données) peuvent compromettre la fonctionnalité du système et la qualité de reproduction. Ce problème peut être abordé typiquement de deux faç ons: un problème d'affectation de ressources, c'est-à-dire comment placer les pairs dans le système et distribuer les données afin d'exploiter plus efficacement les ressources disponibles, et un problème d'incitation (ou de contrôle d'accès), c'est-à-dire comment décourager ou éliminer du système les nœuds qui ne contribuent pas suffisamment.

Adaptation à la topologie du réseau Un autre problème qu'on rencontre dans les systèmes de streaming P2P dérive du fait que les données sont répliquées par les nœuds suivant un chemin non-optimal par rapport au multicast IP natif, puisque l'application P2P ne peut pas connaître la topologie du réseau sous-jacent. Le fait que la localité des connections ne soit pas prise en compte a des effets négatifs, comme un retard supérieur réception du média et une utilisation redondante des liens du réseau, qui peuvent affecter les performances des applications sensibles au temps tels que le streaming en direct.

Robustesse L'usage de nœuds en tant que éléments fonctionnels du système augmente la probabilité que le service sera perturbé par des changements soudains dans la composition du réseau P2P. Puisque une application de streaming P2P de grande échelle repose presque entièrement sur les utilisateurs ordinaires pendant son fonctionnement, le fournisseur original du service ou du contenu a très peu de contrôle sur la manière dont les données sont distribuées et - par conséquent - sur la qualité perçue par les utilisateurs. Contrairement à l'infrastructure réseau sous-jacent (c'est-à-dire des routeurs, des câbles, etc), qui a une très haute disponibilité et tolérance aux pannes, l'overlay applicatif construit par les nœuds d'un système P2P (c'est-à-dire des processus logiciels tournant sur du matériel contrôlé par les utilisateurs) n'offre pas de telles garanties [14]. Chaque nœud peut, à tout moment, entrer dans le système, quitter le système, mal-fonctionner ou tomber en panne: pour fournir un service sans interruption, le système doit être capable à la fois de limiter les effets des perturbations et de rétablir au plus tôt son fonctionnement correcte.

Simplicité d'utilisation La simplicité d'usage et la liberté d'accès au système de streaming live pour tout émetteur potentiel peuvent être considérées comme des défis techniques supplémentaires: l'application ne devrait pas nécessiter d'une grande quantité de ressources (surtout en termes de bande passante remontante) au niveau du fournisseur du contenu, afin que n'importe quel utilisateur qui se connecte à Internet par un commun accès à haut-débit puisse jouer le rôle de la source. L'application devrait ensuite être facile à configurer et à utiliser, de sorte que les utilisateurs occasionnels ne soient pas découragés a participer. Il est important de souligner que, dans un système où le contenu est généré par les utilisateurs, la simplicité d'accès est l'un des facteurs clé qui participe énormément au succès et à la valeur de l'application pour les usagers.

9.1.2 Contributions

Cette thèse apporte plusieurs contributions. En premier lieu, elle aborde le problème du **streaming live d'un point de vue pratique**. Les exigences des applications P2P pour le streaming live à grande échelle sont présentées et discutées à partir de la **liste des défis** énumérés ci-dessus, en considérant les limitations techniques de la technologie Internet actuelle et leur probable évolution dans le futur immédiat. Une étude détaillée de l'état de l'art du streaming live P2P est ensuite proposée, dont l'objectif est d'évaluer les architectures pour le existantes par rapport à leur capacité de supporter un déploiement à grande échelle sur l'Internet. Notre avis à propos des propriétés désirables des systèmes qui aident à réaliser cet objectif est également présenté.

La deuxième contribution est la **conception de PULSE**, un système P2P pour le streaming live qui satisfait aux propriétés précédentes. PULSE est l'un des premiers systèmes qui est basé sur un réseau maillé non structuré, et introduit un mécanisme basé sur l'incitation (*incentive-based*) pour la sélection des nœuds voisins. En exploitant le regroupement spontané des nœuds selon la disponibilité de ressources (qui apparaît en raison des mécanismes d'incitation) et en profitant de la faible synchronisation entre pairs (à l'aide d'une boucle de rétroaction basée sur les performances de réception), PULSE est capable de fonctionner dans une large gamme de scénarios du monde réel. Les avantages de PULSE sur les systèmes actuels peuvent être résumées ainsi:

- Support pour de hauts niveaux de *churn* (nœuds qui arrivent et quittent le système)
- Support pour des distributions de ressources (et notamment de bande passante remontante) très hétérogènes parmi les pairs
- Utilisation efficiente des ressources du système, en particulier en cas de pénurie
- Capacité de s'adapter rapidement aux changements soudains dans les conditions du réseau et du système
- Adaptation implicite à la topologie du réseau en utilisant des mesures de localité
- Attention à la qualité de la reproduction du flux chez l'usager, avec le but de minimiser la dégradation causée par la variation des conditions dans le système au fil du temps

La troisième contribution est un **ensemble d'outils d'évaluation** empirique pour les systèmes de streaming live P2P non-structures (*data-driven*), avec d'autres outils qui permettent d'évaluer la présence de ressources dans le système et la qualité de l'adaptation à la topologie du réseau sous-jacent par des systèmes adaptatifs comme PULSE. Une courte digression sur les techniques qui sont actuellement utilisées pour décrire le comportement des systèmes data-driven, couvrant l'état de l'art des modèles théoriques et des méthodes empiriques, est également incluse.

La quatrième contribution est la réalisation d'un **simulateur** qui modélise le comportement complexe d'un système PULSE. Ensuite, en nous basant sur les connaissances acquises par l'expérimentation par simulation, on a également implémenté un **prototype software** de nœud PULSE. Ces logiciels ont été utilisés pour améliorer notre compréhension du comportement global de PULSE quand le système opère sous une variété de scénarios de distribution de ressources, d'arrivées et départs des nœuds, et de conditions de réseau.

La cinquième et dernière contribution est **l'analyse qualitative et quantitative de PULSE** basée sur les résultats de simulation et d'émulation. Nous avons vérifié d'abord que les algorithmes de PULSE opèrent comme prévu, et ensuite évalué leur performance dans un large éventail de scénarios “critiques” où les systèmes P2P structurés présents dans la littérature ne seraient pas en mesure de fonctionner. Nous avons concentré notre attention sur l'étude de la qualité de l'overlay maillé généré par PULSE, en fonction de la disponibilité de ressources dans le système et de la distribution de la latence du réseau, et sur l'analyse des caractéristiques moyennes des chemins que les données empruntent dans leur trajet de la source aux nœuds récepteurs.

9.2 PULSE et ses algorithmes

PULSE est un système de distribution de flux multimédias en direct qui utilise un mécanisme innovant pour l'adaptation de l'overlay à la disponibilité réelle de ressources dans le système. Le nom PULSE est un acronyme (en anglais) pour "P2P Unstructured Live Streaming Experiment".

9.2.1 Objectifs

L'objectif concret du système PULSE est de permettre *à tout hôte* connecté à l'Internet de distribuer des flux multimédia en direct et/ou de recevoir des flux publies par d'autres hôtes. Les défis techniques de ce scénario applicatif sont les suivants:

- *Tout hôte* signifie qu'il n'y a **aucune garantie sur la stabilité** des nœuds PULSE. Plusieurs raisons pratiques nous incitent à considérer les pairs comme instables, et qui rendent nécessaire la conception d'un système qui tolère les défaillances: les hôtes Internet sont en général peu fiables, car ils peuvent planter ou mal-fonctionner à cause de bogues logiciels ou des actions des utilisateurs; ensuite, un nœud P2P est un logiciel qui peut être activé et arrêté par l'utilisateur à tout moment; enfin, les utilisateurs d'un système live streaming doivent pouvoir interagir avec l'application, par exemple pour faire du "zapping" ou pour s'associer à une autre session de streaming, ce qui peut se produire de manière imprévisible. Nous allons donc supposer que la **durée de vie moyenne** d'un nœud dans le système est **très courte**, dans l'ordre de plusieurs centaines de secondes.

- *Tout hôte* signifie qu'il n'y a **aucune garantie sur la présence de ressources** (notamment en bande passante remontante), ni chez les nœuds prenant part à une session de streaming, *ni chez la source*. Encore une fois pour des raisons pratiques, nous ne voulons pas nous limiter à considérer des scénarios optimistes où tous les nœuds disposent de ressources suffisantes pour recevoir et reproduire au moins une fois le flux: aujourd'hui, alors que la bande passante descendante des accès Internet ADSL commerciaux est désormais suffisante pour obtenir une qualité comparable à la télévision (flux vidéo à partir 500 Kbit/s), la bande passante remontante est souvent beaucoup plus faible, et généralement inférieure à 500 Kbps. Bien que la vitesse moyenne d'accès à Internet (montante aussi bien que descendante) n'ait cessé d'augmenter pendant les dix dernières années, et bien que cette tendance soit destinée a continuer dans le futur, on remarque aussi une augmentation parallèle de la qualité vidéo et du débit [57]. Par conséquent, dans une perspective à moyen terme, nous continuons de penser que le goulot d'étranglement pour la majorité des nœuds sera toujours situé au niveau de la liaison remontante.

- *Tout hôte* signifie qu'il n'y a **aucun mécanisme centralisé de contrôle d'accès** qui puisse vérifier les performances d'un nœud et décider s'il faut autoriser ou refuser son accès au système. Cependant, chaque nœud sera libre d'accéder au système, mais pour que le fonctionnement soit garanti le système dans son ensemble doit se charger de l'équilibrage de la charge et de la répartition des ressources disponibles. Le système devra donc travailler en **modalité “best-effort”**, favorisant une dégradation graduelle des performances en cas de pénurie généralisée de ressources.

Tel est l’environnement dans lequel notre système est censé opérer: un environnement très dynamique, avec des nœuds qui arrivent et partent sans cesse (*churn*), avec des pics d’arrivées et de départs importants (*flash-crowds*), où les noeuds ont un lien d’accès asymétrique, où la bande passante remontante disponible aux nœuds n’est pas uniforme mais largement variable (et souvent bien inférieure à la bande passante du flux), où les nœuds sont considérés comme non-coopératifs par défaut. Un autre contrainte que nous voulons introduire (peut-être moins critique pour l’application en elle-même, mais dont l’impact sur la conception globale du système est très fort), c’est le fait que les algorithmes et le code de PULSE devraient être rendus publiques et pourraient être modifies par les usagers (p. ex. comme le logiciel Bit-Torrent, dont plusieurs implémentations avec d’importantes différences sont disponibles).

9.2.2 Principes et innovations

L’intuition initiale qui a motivé notre travail sur PULSE est qu’il il est possible organiser les membres d’une session de streaming en direct d’une meilleure façon qu’en utilisant des arbres figés: "si nous acceptions de briser cette structure rigide, il serait peut-être possible d’utiliser davantage les informations localement disponibles afin d’optimiser l’overlay pour la distribution des données”. Ce genre d’optimisation n’est pas envisageable avec des arbres traditionnels, car le mantien de leur structure rend tout changement dans l’organisation des nœuds très coûteux.

Aussi, l’asymétrie entre les bandes passantes en upload et download et la pénurie de bande passante remontante (due aux limitations techniques et / ou à la décision explicite par l’usager de ne pas contribuer ses ressources) constituent un problème fondamental qui doit être pris en compte dès le début, même avant de définir la façon dont les nœuds peuvent interagir et choisir leurs partenaires: en effet, un système dans lequel la coopération entre les utilisateurs et la disponibilité de ressources sont postulées a priori et non pas activement poursuivies ne pourrait guère fonctionner dans un environnement tel que l’Internet.

Enfin, nous devons prendre acte du fait que le churn est une propriété intrinsèque au streaming en direct: les utilisateurs peuvent arriver, partir, changer d’une chaîne de streaming à une autre, et même “zapper” rapidement à travers de plusieurs chaînes. L’utilisation d’un overlay statique et figé, qui doit être réparé activement en réponse à tout changement généré par l’activité ordinaire des usagers du système, nous paraissait comme un choix inapproprié, beaucoup moins efficace que de renoncer à la notion classique d’overlay qui était omniprésent dans la littérature.

Ces intuitions nous ont motivés à consacrer nos efforts sur un système non-structuré (*data-driven*), dynamique et utilisant des mécanismes d’incitation au partage. Cependant, on a introduit depuis le début du projet PULSE en 2004 [84] plusieurs innovations par rapport à l’état de l’art:

- L'utilisation d'un **overlay maillé non-structuré** dans un système de streaming live, prenant les distances d'une approche basée sur les arbres (simples ou multiples) qui était considérée comme la seule viable. Les premiers études publiés avec des résultats en support des systèmes non-structurés ont été DONET / Coolstreaming [122] (2005) et Chainsaw [81] (2005).

- Le choix délibéré - opéré à tous les niveaux - d'une **organisation dynamique de l'overlay**. Ce choix a de très profondes répercussions sur le comportement d'un système en présence de churn. Les systèmes structurés considèrent les départs des nœuds comme des événements exceptionnels, et doivent effectuer des opérations *extraordinaires* pour ramener le système dans un nouvel état stable. Un système dynamique, d'autre part, implique qu'une certaine quantité de départs et d'instabilité sera toujours présente même en un régime de fonctionnement normal: les algorithmes contribuent aussi à augmenter le caractère aléatoire, et sont paradoxalement en mesure de l'exploiter pour améliorer la stabilité générale du système. Le premier système publié (à notre connaissance) qui prend en compte le dynamisme produit par le churn comme un aspect du fonctionnement ordinaire du système a été Chunkyspread [113][112](2006).

- L'utilisation de **mesures incitatives pour décourager le freeloading**. Alors que les systèmes basés sur des régimes d'incitation ont reçu beaucoup d'attention ces dernières années, principalement attribuable au succès du système de distribution de contenus BitTorrent [33], l'application de mesures incitatives contre le freeloading dans le contexte du streaming live a été lente et pas très réussie. Plusieurs systèmes ont été conçus qui incluent quelque sorte de mécanisme d'incitation: par exemple, [89] (2004) décrit une architecture multiple-tree où un schéma de tit-for-tat a été mis en œuvre pour prévenir le freeloading. Le problème principal de cet approche est que, puisque les échanges entre nœuds organisés en plusieurs arbres disjoints ne peuvent pas être toujours réciproques, il demande la présence d'un système extérieur (décentralisé!) qui tienne la comptabilité des crédits et des dettes de tous les nœuds du système: cela ajoute une ulterieure couche de complexité au dessus d'un système de streaming.

- L'utilisation de **mesures d'incitation à la contribution des ressources** par les utilisateurs du système. Le plus souvent, lorsque les mesures d'incitation ont été mises en œuvre, leur objectif était de prévenir le freeloading en sanctionnant les nœuds qui contribuent moins que leur quota prévue. Ce n'est que assez récemment, dans [107] (2006), un système multi-arbre est décrit où les nœuds sont autorisés à adhérer à un certain nombre d'arbres qui est proportionnel à leur contribution de bande passante, tandis que les ressources en excès sont allouées à l'ensemble des pairs sans restrictions. Un autre système à arbre multiple, CROSSFLUX [97] (2006), utilise des mécanismes d'incitation pour créer un nombre variable de connexions de backup: les pairs qui contribuent leur juste quantité de ressources sont récompensés par une meilleure protection contre le churn. Dans le contexte des systèmes non structurés, Chunkyspread [112] (2006) prévoit la possibilité d'utiliser des incitations pour biaiser la procédure de sélection des voisins en la faveur des nœuds qui contribuent davantage.

- L’utilisation d’une **boucle de rétroaction** basée sur **l’état présent du progrès de la distribution des données** et sur **des incitations locales** pour obtenir à la fois une adaptation dynamique à l’hétérogeneité des capacités d’upload et à la topologie du réseau. L’approche que nous proposons dans PULSE vise à exploiter les propriétés d’un réseau sous-jacent où les ressources sont inégalement réparties: lorsqu’ils sont placés à proximité de la source, les nœuds capables de contribuer de la capacité en excès peuvent réduire considérablement le retard de reception (*lag*) perçu dans l’ensemble du système, comme si la capacité de service de la source était plus grande que ce qu’elle n’est réellement. D’autre part, les nœuds qui contribuent moins que le débit du flux seront toujours en mesure de participer à la session de streaming. C’est le mécanisme essentiel qui rend la topologie de PULSE adaptative à la distribution des ressources dans le réseau. Aucun des autres systèmes dans la littérature n’utilise à présent cette technique.

Contextuellement à la définition de l’architecture de PULSE, nous avons également introduit une série de techniques de mesure basées sur des paramètres tels que le lag des buffers des nœuds (décrites dans le 4$^{\text{ème}}$Chapitre) afin de *i)* permettre aux nœuds l’acquisition d’informations à propos de l’état de leurs voisins et *ii)* pouvoir analyser et décrire de façon synthetique l’état global d’un système de streaming live. L’utilisation de techniques similaires à celles qu’on a devisé a paru pour la première fois dans une étude sur les performances de PPlive [55] en 2006.

9.2.3 Fonctionnement du système

Les nœuds rejoignent le système en contactant un nœud quelconque qui fait déjà partie de la session PULSE. La façon dont cette information est obtenue peut être adaptée selon les exigences de l’application: à présent, le point d’entrée du système est spécifié dans le fichier de configuration de chaque session PULSE (fichier *.pulse*, dont la fonction et le contenu sont similaires aux fichiers *.torrent* de BitTorrent).

Control Plane Les nœuds échangent deux types de messages, appelés RED et BLUE. Les messages BLUE contiennent des informations sommaires sur la disponibilité de données chez un nœud sous la forme de l’intervalle de lag des extremités de son buffer de reception. Les messages RED contiennent une description détaillée du contenu d’une partie du buffer de chaque noeud (la *Trading Window*, voir le Chapitre 4, Fig. 3.1) qui est encodée sous forme de bitmap. Suivant la quantité d’information disponible au sujet des nœuds voisins, ceux-ci sont organisés dans deux listes (*Blue Knowledge List*, *Red Knowledge List*). Les nœuds qui font partie de la *Red Knowledge List* peuvent être choisis pour effectuer des échanges de données.

Data Plane Les échanges de données entre nœuds sont déterminés par deux algorithmes fondamentaux: l’algorithme de selection des nœuds (*peer selection*) et l’algorithme de selection des pièces (*chunk selection*). Sans trop déscendre dans les détails, la selection des nœuds est basée sur des critères de rétribution (*incentives*) et sur la disponibilité de pièces utiles (*chunk availability*). Les deux mécanismes principaux pour le choix des nœuds sont la MISSING selection, qui permet les échanges réciproques entre nœuds partageant un interêt pour une même sequence de pièces du stream, et la FORWARD selection, qui permet l'exercice de l'altruisme entre nœuds qui ne partagent pas d'interêt réciproque. Le critère de selection des pièces est basé sur la rareté des chunks parmi les voisins de la *Red Knowledge List* de chaque nœud: périodiquement, tous les pairs envoient des requêtes pour un ou plusieurs chunks à leur voisins utilisant un critère *rarest-first*; ensuite, les nœuds qui reçoivent les requêtes choisissent quels chunks seront envoyés à quel voisin, utilisant un critère *least-sent-first* qui vise à rendre homogène la réplication des chunks (evitant, par example, que les chunks les plus rares soient envoyés trop souvent par un même nœud à dépit des autres chunks).

9.2.4 Implémentation

Une première version fonctionnelle du prototype du nœud PULSE a été mise au point pendant l'été 2006 avec l'aide de Diego Perino, qui a écrit un compte rendu détaillé de son activité dans son rapport de stage [82]. Le noeud a été mis en œuvre prenant comme référence le code du simulateur PULSE pour les algorithmes d'échange de données, tandis que la plupart des structures de données ont dû être adaptées et l'interface au réseau a dû être implementée. L'organisation interne du nœud est décrite dans le 3ème chapitre.

Le prototype du nœud PULSE est écrit en Python, un langage de programmation et scripting orienté-objet qui est bien adaptée pour développer rapidement des applications simples. L'un des principaux avantages d'utiliser Python pour notre logiciel est la disponibilité du framework Twisted, qui offre plusieurs fonctions de bas niveau utiles pour une application de réseau (accès au sockets en multiplexing, gestion des buffers entre application et réseau, etc.). Le prototype utilise également d'autres modules externes, notamment un wrapper Python pour l'implémentation des codes FEC Reed-Solomon en C++ crée par Luigi Rizzo [93]. En outre, le nœud PULSE intègre une implémentation du protocole SCAMP tel qu'il est défini dans [47] (à l'exception du mécanisme d'indirection). Quelques autres modifications mineures (comme un champ ID de message) ont été nécessaires pour limiter l'envoi de messages redondants dans des déployements de petite taille, où l'utilisation de *random walks* de longueur variable (avec des critères de terminaison probabilistes) produisait une grande quantité de trafic réseau à chaque fois qu'un nouveau nœud se joignait à une session PULSE.

Le nœud PULSE utilise deux sockets dans son fonctionnement: un socket TCP pour les transferts de données et un socket UDP pour l'échange de messages de contrôle (RED et BLUE). Le choix des connexions TCP pour les transferts de données est motivé par le besoin de fiabilité (*reliability*), car les données des chunks vidéo s'étalent typiquement sur plusieurs datagrammes IP, ce qui rend le protocole de transport UDP non approprié à la tâche. PULSE peut en effet utiliser des tailles de chunks assez larges, et des débits de génération assez faibles, grâce aux contraintes de temps moins strictes du streaming live par rapport à la distribution interactive de la vidéo. Par exemple, la taille typique d'un chunk pourra aller de quelques à plusieurs dizaines de kilo-octets, alors que des débits raisonnables vont de 2 à 16 chunks (environ) par seconde. Nous rappelons que, une fois le débit fixé, la taille des chunks dépend du débit du stream diffusé et peut donc changer d'une session PULSE à l'autre. L'utilisation d'UDP pour les messages de contrôle est motivée par le fait que la perte occasionale de ces messages est tolérable et n'impacte pas le fonctionnement du système. La taille des messages de contrôle UDP dépend principalement du nombre de chunks dans la Trading Window du buffer (qui détermine la taille de la bitmap), et est généralement inférieure à 100 octets.

9.3 Évaluation

La partie la plus importante de cette thèse est dédiée à l'évaluation du comportement du système PULSE. Tout d'abord, le premier problème auquel nous avons été confrontés a été le manque d'un cadre théorique bien défini pour l'évaluation des overlays non-structurés. Le Chapitre 5 contient une analyse des caracteristiques des systèmes data-driven et propose une série de *metrics* qui peuvent être utilisés dans un cas général. Ensuite, nous élaborons des metrics originales pour étudier spécifiquement PULSE, qui permettent de mettre en relation la présence de ressources aux nœuds avec leur comportement et leur performance de de réception du flux.

Le deuxième problème à affronter était la question fondamentale: est-ce que les algorithmes de PULSE peuvent réellement fonctionner? Et, dans ce cas, quelles sont les propriétés globales du système? Pour repondre à ces questions, nous avons choisi de mettre en œuvre une analyse basée sur la simulation et sur l'émulation d'un réseau PULSE. Le Chapitre 6 contient la description des methodes de simulation utilisées, leurs limitations et avantages, ainsi que les résultats de simulation que nous avons obtenu. Ensuite, le Chapitre 7 décrit les deployements expérimentaux du prototype de nœud PULSE que nous avons réalisés en utilisant des testbeds à large échelle comme Grid'5000 [3] et PlanetLab [5].

9.3.1 Propriétés des systèmes non-structurés

D'un point de vue analytique, les systèmes data-driven sont plus difficiles à décrire et à évaluer que les systèmes structurés classiques. Les raisons sont les suivantes:

1. Les données ne suivent pas le même chemin sur l'overlay, ni la trajectoire d'un morceau de données est prévisible, même avec une connaissance complète de l'état de tous les nœuds dans le réseau.

2. Des mécanismes locaux de réconciliation des données sont généralement utilisés pour éviter tout risque de duplication redondante. La réconciliation nécessite l'échange d'informations de contrôle qui décrivent les chunks contenus dans le buffer de chaque noeud. Tenir en compte l'état de la connaissance que chaque nœud a du reste du système pour expliquer les décisions que les nœuds prennent est très difficile si on veut observer le réseau d'une perspective globale.

3. Les connexions ouvertes entre les nœuds peuvent transporter seulement des messages de contrôle, ou bien soit les données soit les informations de contrôle. L'existence d'une connexion entre deux nœuds n'implique donc pas que des données soient régulièrement échangées dans les deux sens.

4. L'overlay évolue au fil du temps puisque les liens qui le constituent sont renégociés localement parmi les nœuds.

PULSE possède toutes les propriétés d'un système data-driven. En plus, la topologie de son overlay évolue très rapidement et de façon chaotique dans le temps, puisque elle est influencée par la rétroaction (*feedback*) basée sur l'état des échanges de données entre nœuds. Ce fait nous a poussé à utiliser un approche empirique (plutôt que théorique) dans l'étude des systèmes data-driven et de PULSE.

9.3.2 Simulation

La simulation des algorithmes de PULSE a été la première étape de notre analyse. Après avoir décidé d'implementer un simulateur 'maison' pour décrire le système avec un niveau de détail approprié (le raisons de ce choix sont expliquées dans le Chapitre 4), on a opté pour une approche utilisant une division du temps en unités discrètes (*time-slots* ou *simulation steps*).

Pour simplifier les choses, on a introduit une synchronisation partielle des transferts de données dans l'ensemble du système, afin qu'ils respectent toujours les limites temporelles d'un *step* de simulation. Cela peut être implementé (comme dans notre cas) en limitant le nombre total de morceaux qu'un nœud peut générer au cours de chaque time-slot. Le montant de bande passante en upload / download détermine ainsi le nombre de *data chunks* qui peuvent être échangés. Avec cette technique, nous pouvons être sûrs que tous les transferts de chunks seront achevés avec un délai maximal d'un time-step (si la capacité totale est utilisée) ou avant la fin du step (si la capacité n'est pas epuisée), mais en aucun cas après cette limite.

La simplification ci-dessus nous amène vers le concept de *bandwidth slot*, qui constitue l'unité de mesure pratique pour représenter les transferts de données dans notre modèle. Un *bandwidth slot* est défini comme la quantité de bande passante nécessaire pour transmettre un chunk dans un time-slot. Les capacités des nœuds peuvent alors être définies en termes de multiples de bandwidth slots: cette quantisation de la bande passante peut approximer assez bien le comportement typique de TCP en cas de congestion au niveu du bord du réseau, car la capacité d'upload disponible est partagée équitablement par tous les transferts de chunks qui se déroulent au même temps.

Au sujet des latences de propagation introduites par le réseau, la durée relativement large que nous avons choisie pour notre time-step a des effets positifs. Si nous considérons les valeurs typiques de latence entre deux nœuds mesurée sur l'Internet, qui vont de quelques dizaines à quelques centaines de millisecondes, nous pouvons approximer par excès le délai nécessaire pour qu'un message parvienne à un autre nœud avec la durée d'un seul step. Cette possibilité simplifie le modele d'échange des messages de contrôle, qui sont assez courts (quelques dizaines d'octets), et dont le délai de transfert est dominé par la latence. Représenter la latence de réseau par un délai constant d'un seul time-step nous aide à mieux comprendre l'évolution de l'état interne du système, puisque les changements d'état induits par les messages de contrôle prennent effect avec un retard uniforme.

Les choix que nous avons faits lors de la définition de notre modèle du système PULSE cherchent à produire une description simplifiée mais fidèle de la dynamique interne et du comportement global du système à l'état d'équilibre. Il y a évidemment des limites entrainées par ce compromis entre simplicité et fidélité sur la portée des résultats qu'on peut obtenir en utilisant notre approche: ces limites sont examinées en détail dans la Section 6.1.

9.3.3 Émulation sur large échelle

La deuxième étape de notre analyse nous a amenés à valider les resultats produits par les simulations en réalisant l'émulation d'un système PULSE dans un environnement contrôlé. Utilisant le nœud prototype sur des testbeds à large échelle, on a pu largement confirmer que le comportement global d'un système "réel" est bien représenté par le modèle que notre simulateur implémente.

Après la phase de validation, on a mis en œuvre un déployement de moyenne échelle sur PlanetLab, un testbed composé de machines connectées à Internet localisées dans le monde entier: ces conditions sont assez répresentatives de l'environnement réel qu'une application distribuée rencontre lors de son déploiement. Ce testbed nous a permis d'apprécier de façon préliminaire l'impact sur le comportement de PULSE dû à la présence d'une latence non uniforme entre les nœuds. On a aussi confirmé que l'utilisation de la latence en tant que critère sécondaire pour la *peer selection* peut avoir un rôle important dans l'optimisation de la qualité globale de l'overlay.

9.3.4 Résultats de l'évaluation

Nous avons évalué le comportement du système PULSE dans un grand nombre de scenarios, en observant les effets du changement des paramètres suivantes (parmi les autres):

- Disponibilité totale de bande passante dans le système par rapport à la demande des nœuds. Ce paramètre, introduit en [27], est connu sous le nom de *Resource Index* (abregé en *RI*) et permet de quantifier la présence de ressources en excès.

- Distribution de la bande passante en upload des nœuds. Nous avons utilisé des scénarios avec un nombre variable de "classes" de bande passante, avec une distribution des capacités d'accès plus ou moins hétérogène.

- Taille du système et arrivées / départs de nœuds intempestifs (*churn*)

Les résultats des simulations nous ont permis d'apprécier les phénomènes suivants:

1. Le système réjoint rapidement un équilibre stable en présence de valeurs de $RI > 1$, même si l'excès de bande passante est assez faible et/ou en présence d'hétérogénéité.

2. La valeur de RI détermine les performances du système après convergence. En présence de ressources en excès, la distance moyenne en hops des nœuds par rapport à la source est réduite, avec un gain en termes de délai moyen de réception des chunks (*node lag*).

3. Les nœuds qui appartiennent à des classes de bande passante différentes s'attestent sur des positions différentes dans l'overlay: une corrélation très forte est présente entre proximité à la source et disponibilité d'upload, avec les nœuds les plus "riches" placés en moyenne plus près que les "pauvres". Cette corrélation tend à faiblir avec l'augmentation de la valeur de RI.

4. La présence d'hétérogénéité résulte en un overlay plus compacte, c'est à dire où les nœuds ont une distance moyenne de la source inférieure, par rapport à un scénario homogène avec le même RI.

5. La dépendence entre la taille du système et le *node lag* moyen de ses participants est logarithmique, à condition que la disponibilité de ressources soit suffisante (toujours $RI > 1$, les scénarios homogènes démandant un surplus de ressources légèrement plus élévé).

6. Le système offre une très bonne résilience aux changements soudains (départs et arrivées de nœuds), même quand ces changements intéressent une large partie des nœuds présents.

On a ensuite analysé attentivement les trajectoires des chunks à l'intérieur du système et la contribution de chacun des différents aspects des algorithmes de sélection des nœuds voisins (Chapitre 6). Les points suivants offrent une synthèse des résultats acquis au cours de notre travail sur PULSE:

Le tit-for-tat est efficace pour optimiser la réception Nos résultats confirment que l'utilisation d'un mécanisme d'incitation réciproque tel que le tit-for tat, combiné à une rétroaction basée sur les performances de réception (*node lag*), fournit un mécanisme puissant pour l'optimisation de la qualité de l'overlay. Sans aucune connaissance *a priori* quant à la disponibilité de ressources des autres noeuds, et en dépit de la courte durée moyenne des intéractions entre nœuds, chaque peer atteint et maintient une position assez stable à l'intérieur du système, ce qui détermine à son tour la réception régulière du flux de données. L'analyse des chemins de distribution des données dans le système à l'équilibre indique que les nœuds des classes les plus "riches" en bande passante ont une haute chance d'être traversés très tôt au cours de la distribution de chaque chunk.

Le *node lag* est efficace comme mécanisme de discrimination Si l'on étudie un système PULSE en pénurie de ressources, on constate que les nœuds qui contribuent le plus ne souffrent que très peu souvent de la dégradation des performances. Notre conclusion est que l'utilisation du *node lag* des voisins en tant que facteur discriminant pour la sélection des nœuds avec qui coopérer permet au système de résister à l'influence negative du *freeloading* (contribution de ressources nulle ou inférieure au débit du flux).

L'altruisme a un rôle essentiel La présence d'altruisme a clairement une grande importance, car il peut améliorer l'efficace de l'incitation par rapport à l'usage d'un mécanisme de "tit-for-tat pur". Nos simulations montrent que, par l'ajout de juste quelques connexions vers des nœuds qui ne seraient autrement pas choisis en base au critère "tit-for-tat pur", les performances et la stabilité globale du système s'améliorent de manière remarquable. La meilleure utilisation de l'upload assurée par l'altruisme favorise une diffusion plus rapide des chunks au niveau des toutes premières duplications (qui sont effectuées, on le rappelle, par le nœuds les plus riches en ressources). Le résultat global de l'altruisme est une réduction généralisée du délai de réception.

9.4 Conclusions

Les resultats offerts par PULSE montrent que un choix approprié des politiques de sélection de nœuds voisins permet la création d'overlays adaptatifs et dynamiques, capables de fournir un service best-effort dans un large éventail de scénarios. Les conclusions qu'on a pu tirer de notre travail d'analyse empirique nous ont permis de mieux comprendre les avantages des overlays data-driven, qui se revèlent formidables dans le cas d'applications destinées à fonctionner dans des environnements réels avec des conditions aléatoires et non maîtrisables et des nœuds *a priori* non cooperatifs.

Notre méthode, qui propose l'utilisation combinée d'incitations réciproques et d'une boucle de rétroaction basée sur le délai moyen de réception, s'est révelée adéquate par rapport à notre but initial: définir une architecture pour un système de streaming live pouvant opérer sur Internet à une échelle globale. Nous souhaitons que le résultats que nous avons obtenu puissent alimenter l'activité de recherche dans le domaine de la distribution de contenus, surtout pour ce qui concerne l'atteinte de l'efficacité optimale et le support d'environnements non-cooperatifs.

Bibliography

[1] Directive 2001/29/EC of the european parliament and of the council of 22 may 2001 on the harmonisation of certain aspects of copyright and related rights in the information society.

[2] Electronic Frountier Foundation - list of lawsuits against P2P users and service providers - http://www.eff.org/issues/file-sharing.

[3] Grid'5000 - grid platform testbed - http://www.grid5000.org.

[4] Loi n. 2006-961 du 1er août 2006 relative au droit d'auteur et aux droits voisins dans la société de l'information. J.O. n. 178 du 3 août 2006 page 11529.

[5] PlanetLab - http://www.planet-lab.org/.

[6] The Trusted Computing Group - https://www.trustedcomputinggroup.org/.

[7] USA H.R. 2281, DMCA (digital millenium copyright act), oct. 20, 1998.

[8] E. Adar and B. Huberman. Free riding on Gnutella. Technical report, Xerox PARC, 10 Aug. 2000, Aug 2000. First Monday, 5(10), Oct 2000.

[9] A. Ali, A. Mathur, and H. Zhang. Measurement of commercial peer-to-peer live video streaming. In *Proceedings of 1st Workshop in Recent Advances in Peer-to-Peer Streaming*, August 2006.

[10] S. Annapureddy, C. Gkantsidis, and P. Rodriguez. Providing video-on-demand using peer-to-peer networks. In *Internet Protocol TeleVision (IPTV) workshop in conjunction with WWW '06*, May 2006.

[11] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, USA, 1984.

[12] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of the 2002 ACM SIGCOMM Conference*, Aug. 2002.

[13] S. Banerjee, T. G. Griffin, and M. Pias. The interdomain connectivity of PlanetLab nodes. In *Proceedings of PAM, Antibes Juan-les-Pins, France*, April 2004.

[14] M. Bawa, H. Deshpande, and H. Garcia-Molina. Transience of peers and streaming media. In *HotNets-I, Princeton, NJ, USA*, pages 107–112, 2002.

[15] Berkeley/LNBL/ISI. The NS-2 network simulator. http://www.isi.edu/nsnam/ns/, 1989-.

[16] A. Bharambe, S. Rao, V. Padmanabhan, S. Seshan, and H. Zhang. The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems, February 2005*, 2005.

[17] E. W. Biersack. Where is multicast today? *SIGCOMM Comput. Commun. Rev.*, 35(5):83–84, 2005.

[18] A. Brampton, A. Macquire, I. A. Rai, N. Race, L. Mathy, and M. Fry. Characterising user interactivity for sports video-on-demand. In *Proc. of the 17th International Workshop on Network and Operating Systems Support for Digital Audio and Video, (NOSSDAV 2007), Urbana-Champaign, IL, USA*, June 2007.

[19] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in P2P systems. In *Proceedings of the Third IEEE International Conference on Peer-to-Peer Computing (P2P 2003)*, 2003.

[20] D. Carra, R. L. Cigno, and E. W. Biersack. Fast stochastic exploration of p2p file distribution archite ctures. In *Proc. IEEE GLOBECOM'06, San Francisco, USA*, Nov. 2006.

[21] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proceedings of IPTPS'03*, February 2003.

[22] A. Chaintreau, F. Baccelli, and C. Diot. Impact of TCP-like congestion control on the throughput of multicast groups. *IEEE/ACM Trans. Netw.*, 10(4):500–512, 2002.

[23] B. Chang, Y. Shi, and N. Zhang. Refine DONet's overlay with network distance estimation. In *Proc. of the First Workshop on Recent Advances in Peer-to-Peer Streaming (WRAIPS 2006)*, Waterloo, Canada, August 2006.

[24] Y. Chawathe. Scattercast: an adaptable broadcast distribution framework. *Multimedia Syst.*, 9(1):104–118, 2003.

[25] Y. Chawathe, S. McCanne, and E. A. Brewer. RMX: reliable multicast for heterogeneous networks. In *Proceedings of IEEE INFOCOM'00, Tel Aviv, Israel*, pages 795–804, March 2000.

[26] S. Cheshire. Latency and the quest for interactivity. White paper commissioned by Volpe Welty Asset Management, L.L.C., for the Synchronous Person-to-Person Interactive Computing Environments Meeting, San Francisco, November 1996.

[27] Y. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an internet broadcast system based on overlay multicast. In *USENIX Annual Technical Conference, General Track*, pages 155–170, 2004.

[28] Y. H. Chu, J. Chuang, and H. Zhang. A case for taxation in peer-to-peer streaming broadcast. In *Proc. of the ACM SIGCOMM workshop on Practice and theory of Incentives in Networked Systems (PINS'04), Portland, OR, USA*, 2004.

[29] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. 20(8):1456–1471, Oct. 2002.

[30] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics'00, Santa Clara, CA*, pages 1–12, June 2000.

[31] Y.-H. Chu and H. Zhang. Considering altruism in peer-to-peer internet streaming broadcast. In *Proc of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'04), Kinsale, County Cork, Ireland*, June 2004.

[32] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *In Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA*, page 46, 2000.

[33] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems, Berkeley*, June 2003.

[34] G. Dán, V. Fodor, and I. Chatzidrossos. On the performance of multiple-tree-based peer-to-peer live streaming. In *Proceedings of IEEE INFOCOM'07, Anchorage, Alaska*, 2007.

[35] S. E. Deering. Multicast routing in internetworks and extended LANs. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 55–64, New York, NY, USA, 1988. ACM Press.

[36] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over peers. Technical report, Tech. Rep. 2001-31, CS Dept., Stanford University, 2001.

[37] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. The pollution attack in p2p live video streaming: Measurement results and defenses. In *Proceedings of the 2nd P2P-TV Workshop, in conjunction with ACM SIGCOMM 2007, Kyoto, Japan*, 2007.

[38] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network magazine special issue on Multicasting*, 14(1):78–88, January/February 2000.

[39] J. R. Douceur. The sybil attack. In *Proceedings of the IPTPS02 Workshop, Cambridge, MA (USA)*, 2002.

[40] J. Edmonds. Edge-disjoint branchings. *Combinatorial Algorithms, R. Rustin, Ed.*, pages 91–96, 1972.

[41] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A. M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, November 2003.

[42] B. Fan, D.-M. Chiu, and J. C. Lui. The delicate tradeoffs in BitTorrent-like file sharing protocol design. In *Proc. of the International Conference on Network Protocols (ICNP), Santa Barbara, USA*, 2006.

[43] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, Dec. 1997.

[44] P. Francis. Yoid: Extending the internet multicast architecture. Technical report, ACIRI, April 2000.

[45] A.-T. Gai, D. Lebedev, F. Mathieu, F. de Montgolfier, J. Reynier, and L. Viennot. Acyclic preference systems in P2P networks. In *Proceeedings of the 13th European Conference on Parallel and Distributed Computing (Euro-Par 2007)*, 2007.

[46] A.-T. Gai and L. Viennot. PrefixStream: a balanced, resilient and incentive peer-to-peer multicast algorithm. Technical report, Technical Report RR-5514, INRIA Rocquencourt, March 2005.

[47] A. Ganesh, A. M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52:139–258, 2003.

[48] A. J. Ganesh, A. M. Kermarrec, and L. Massoulié. SCAMP: peer-to-peer lightweight membership service for large-scale group communication. In Springer-Verlag, editor, *Proceedings of the Third International COST264 Workshop*, pages 44–55, 2001.

[49] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of IEEE INFOCOM'04, Hong Kong*, 2004.

[50] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, 18(5):74–93, September 2001.

[51] C. Grothoff. An excess-based economic model for resource allocation in peer-to-peer networks. *Wirtschaftsinformatik*, June, 2003.

[52] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[53] D. Hales and S. Arteconi. SLACER: a self-organizing protocol for coordination in peer-to-peer networks. *Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications]*, 21(2):29–35, 2006.

[54] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto. Mapping peer behavior to packet-level details: A framework for packet-level simulation of peer-to-peer systems. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS'03)*, Orlando, USA, 2003.

[55] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. Insights into PPLive: a measurement study of a large-scale P2P IPTV system. In *Proceedings of IPTV Workshop, International World Wide Web Conference*, 2006.

[56] X. Hei, Y. Liu, and K. Ross. Inferring network-wide quality in P2P live streaming systems. *IEEE JSAC,* Special Issue on Advances in P2P Streaming:to be published, 2007.

[57] C. Huang, J. Li, and K. Ross. Can Internet VoD be profitable? In *Proceedings of ACM SIGCOMM 2007, Kyoto, Japan*, 2007.

[58] C. Huitema. The case for packet level FEC. In *Proceedings of IFIP 5th Int'l Workshop on Protocols for High Speed Networks, Sophia Antipolis, France,* October 1996.

[59] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, October 2000.

[60] M. Jelásity, A. Montresor, G. P. Jesi, and S. Voulgaris. Peersim peer-to-peer simulator - http://peersim.sourceforge.net/, 2004.

[61] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is P2P dying or just hiding? In *IEEE Global Telecommunications Conference*, volume 3, pages 1532–1538, 2004.

[62] R. Karp, A. Sahay, E. Santos, and K. Schauser. Optimal broadcast and summation problem in the LogP model. In *In Proc of ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 142– 153, 1993.

[63] J. Keller and G. Simon. SOLIPSIS: a massively multi-participant virtual world. In *International Conference on Parallel and Distributed Techniques and Applications*, 2003.

[64] S. Keshav. REAL network simulator - http://www.cs.cornell.edu/skeshav/real/overview.html, 1989-1997.

[65] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of ACM SOSP, 2003*, 2003.

[66] R. Kumar, Y. Liu, and K. W. Ross. Stochastic fluid theory for P2P streaming systems. In *Proceedings of IEEE INFOCOM'07, Anchorage, Alaska*, 2007.

[67] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in BitTorrent systems. In *Proceedings of ACM SIGMETRICS'2007, San Diego, CA, USA*, June 2007.

[68] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free riding in BitTorrent is cheap. In *Fifth Workshop on Hot Topics in Networks (HotNets-V), Irvine, CA, USA*, Nov. 2006.

[69] N. Magharei and R. Rejaie. Understanding mesh-based peer-to-peer streaming. In *Proceedings of ACM NOSSDAV'06, Newport, Rhode Island, USA*, May 2006.

[70] N. Magharei and R. Rejaie. PRIME: peer-to-peer receiver-driven mesh-based streaming. In *Proceedings of IEEE INFOCOM'07, Anchorage, Alaska*, 2007.

[71] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live P2P streaming approaches. In *Proceedings of IEEE INFOCOM'07, Anchorage, Alaska*, 2007.

[72] L. Massoulié, A. Twigg, C. Gkantsidis, and P. R. Rodriguez. Randomized decentralized broadcasting algorithms. In *Proceedings of IEEE INFOCOM'07, Anchorage, Alaska*, 2007.

[73] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 53–65, Mar. 2002.

[74] T. Moreton and A. Twigg. Trading in trust, tokens, and stamps. In *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[75] G. Neglia, G. L. Presti, H. Zhang, and D. Towsley. A network formation game approach to study BitTorrent Tit-for-Tat. In *Proceedings of EuroFGI International Conference on Network Control and Optimization*, June 2007.

[76] S. R. Network. SSFNET network simulator - http://www.ssfnet.org/homepage.html, 1999.

[77] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings of IEEE INFOCOM'02, New York, NY, USA*, pages 170–179, June 2002.

[78] T. Nguyen and A. Zakhor. Distributed video streaming with forward error correction. In *Proceedings of Packet Video Workshop, Pittsburgh, USA*, 2002.

[79] J. Nonnenmacher and E. W. Biersack. Scalable feedback for large groups. *IEEE/ACM Trans. Netw.*, 7(3):375–386, 1999.

[80] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 177–186, 2002.

[81] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems, February 2005*, 2005.

[82] D. Perino. The PULSE system. A new P2P prototype for live streaming. Master's thesis, Institut Eurecom, Sophia-Antipolis, France, 2006.

[83] R. Peterson and E. G. Sirer. Going beyond tit-for-tat: Designing peer-to-peer protocols for the common good. In *Proceedings of the Workshop on Future Directions in Distributed Computing, Bertinoro, Italy*, June 2007.

[84] F. Pianese. P2P live media streaming: Delivering data streams to massive audiences within strict timing constraints. Master's thesis, Institut Eurecom, Sophia-Antipolis, France, 2004.

[85] F. Pianese, J. Keller, and E. W. Biersack. PULSE, a flexible P2P live streaming system. In *Proceedings of the 9th IEEE Global Internet Symposium 2006, in conjunction with IEEE Infocom 2006, Barcelona, Spain*, Apr. 2006.

[86] F. Pianese and D. Perino. Resource and locality awareness in an incentive-based P2P live streaming system. In *Proceedings of the 2nd P2P-TV Workshop, in conjunction with ACM SIGCOMM 2007, Kyoto, Japan*, August 2007.

[87] F. Pianese, D. Perino, J. Keller, and E. W. Biersack. PULSE: an adaptive, incentive-based, unstructured P2P live streaming system. *IEEE Transactions on Multimedia*, November 2007. Special Issue on Content Storage and Delivery in Peer-to-Peer Networks.

[88] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *Proc. of the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007)*, Apr. 2007.

[89] J. Pouwelse, J. Taal, R. Lagendijk, D. Epema, and H. Sips. Real-time video delivery using peer-to-peer bartering networks and multiple description coding. In *Proceedings of the IEEE Int'l Conference on Systems, Man and Cybernetics*, October 2004.

[90] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-peer networks. In *Proceedings of the 2004 ACM SIGCOMM Conference, Portland, OR, USA*, 2004.

[91] J. Ritter. Why gnutella can't scale. no, really. http://www.monkey.org/ dug-song/mirror/gnutella.html, February 2001.

[92] L. Rizzo. Dummynet: A simple appproach to the evaluation of network protocols. *Computer Communication Review*, 27(1):31–41, Jan. 1997.

[93] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *Computer Communication Review*, 27(2):24–36, April 1997.

[94] A. Rodriguez, C. Killian, S. Bhat, D. Kostić, and A. Vahdat. MACEDON: methodology for automatically creating, evaluating, and designing overlay networks. In *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[95] S. Sanghavi, B. Hajek, and L. Massoulié. Gossiping with multiple messages. *IEEE Transactions on Information Theory*, (12), Dec. 2007.

[96] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking*, 2002.

[97] M. Schiely and P. Felber. CROSSFLUX: an architecture for peer-to-peer media streaming. *Global Data Management, Volume 8, Emerging Communication: Studies on New Technologies and Practices in Communication, IOSPress*, 8:342–358, 2006.

[98] M. Schiely, L. Renfer, and P. Felber. Self-organization in cooperative content distribution networks. In *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA'05), Cambridge, MA*, 2005.

[99] S. Sheu, K. A. Hua, and W. Tavanapong. Chaining: A generalized batching technique for video-on-demand. In *ICMCS*, pages 110–117, 1997.

[100] J. Shneidman, D. Parkes, and L. Massoulié. Faithfulness in internet algorithms. In *Proc. of the ACM SIGCOMM workshop on Practice and theory of Incentives in Networked Systems (PINS'04), Portland, OR, USA*, 2004.

[101] J. Shneidman and D. C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. 23rd ACM Symp. on Principles of Distributed Computing (PODC'04), St. John's, Canada*, July 2004.

[102] T. Silverston and O. Fourmaux. Measuring P2P IPTV systems. In *Proceedings of ACM NOSSDAV'07, Urbana-Champaign, IL, USA*, June 2007.

[103] T. Small, B. Liang, and B. Li. Scaling laws and tradeoffs in peer-to-peer live multimedia streaming. In *in Proc. of ACM MM'06, October 23-27, 2006, Santa Barbara, California, USA.*, 2006.

[104] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proceedings of the 2004 ACM SIGCOMM Conference, Portland, OR, USA*, 2004.

[105] M. Steiner, E. W. Biersack, and T. En-Najjary. Actively monitoring peers in kad. In *Proceedings of the* 6^{th} *International Workshop on Peer-to-Peer Systems (IPTPS'07)*, 2007.

[106] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[107] Y.-W. Sung, M. Bishop, and S. G. Rao. Enabling contribution awareness in an overlay broadcasting system. In *Proceedings of the 2006 ACM SIGCOMM Conference, Pisa, Italy*, pages 411–422, 2006.

[108] K. Tamilmani, V. Pai, and A. E. Mohr. SWIFT: a system with incentives for trading. In *Proceedings of the 2nd Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[109] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. BubbleStorm: resilient, probabilistic, and exhaustive Peer-to-Peer search. In *Proceedings of ACM SIGCOMM 2007, Kyoto, Japan*, Aug. 2007.

[110] D. A. Tran, K. A. Hua, and T. T. Do. A peer-to-peer architecture for media streaming. *IEEE JSAC Special Issue on Advances in Service Overlay Networks*, 22(1, Jan. 2004):121–133, 2004.

[111] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostič, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of the 5th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI), Boston, MA*, 2002.

[112] V. Venkataraman, P. Francis, and J. Calandrino. Chunkyspread: Multi-tree unstructured peer-to-peer multicast. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems, February 2006*, 2006.

[113] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured end system multicast. In *Proceedings of the 14th IEEE International Conference on Network Protocols*, 2006.

[114] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: a secure economic framework for peer-to-peer resource sharing. In *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[115] V. Vishnumurthy and P. Francis. On overlay construction and random node selection in heterogeneous unstructured P2P networks. In *Proceedings of IEEE INFOCOM'06, Barcelona, Spain*, 2006.

[116] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: enhancing bittorrent for supporting streaming applications. In *9th IEEE Global Internet Symposium 2006 (in Conjunction with IEEE INFOCOM 2006), Barcelona, Spain*, 2006.

[117] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217, Jun 2005.

[118] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of OSDI '02*, December 2002.

[119] W. Yang and N. Abu-Ghazaleh. GPS: a general peer-to-peer simulator and its use for modeling BitTorrent. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, pages 425 – 434, 2005.

[120] X. Yang and G. de Veciana. Service capacity of peer-to-peer networks. In *Proceedings of IEEE INFOCOM'04, Hong Kong,* Mar. 2004.

[121] 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 *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 287–290, New York, NY, USA, 2005. ACM Press.

[122] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of IEEE INFOCOM'05, Miami, FL, USA*, March 2005.