DELAY-CENTRIC APPROACH FOR PEER-TO-PEER VIDEO LIVE STREAMING

Anis Ouali

A thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy

Concordia University

Montréal, Québec, Canada

June 2011

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mr. Anis Ouali**

Entitled: **Delay-Centric Approach for Peer-to-Peer Video Live Streaming**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. Rachida Dssouli _____ Chair

Dr. Jiangchuan Liu _____ External Examiner

Dr. Anjali Agarwal _____ Examiner

Dr. Dongyu Qiu _____ Examiner

Dr. Lata Narayanan _____ Examiner

Dr. Brigitte Jaumard _____ Supervisor

Dr. Brigitte Kerhervé _____ Supervisor

Approved _____

Chair of Department or Graduate Program Director

Abstract

Delay-Centric Approach for Peer-to-Peer Video Live Streaming

Anis Ouali, Ph.D.

Concordia University, 2011

Peer-to-peer (P2P) systems are quite attractive due to their ability to deliver large amounts of data at a reduced deployment cost. They offer an interesting paradigm for media streaming applications that can benefit from the inherent self organization and resource scalability available in P2P systems.

Recently, some push-pull scheduling strategies have been proposed to replace the classical pull mechanism in mesh based P2P streaming systems. A push-pull mechanism is more efficient in terms of the overheads and leads to much better playback delay performance since the pull part is mainly used either at the beginning of the session or to recover missing content.

In order to exploit such an advantage, we propose to revisit peering strategies based on the use of a push-pull content retrieval mechanism. Our focus is to minimize the playback delay experienced by participating nodes. We propose two new peering strategies that we compare to the state-of-the-art strategies using simulation.

To validate the overlay construction results obtained by simulation, we propose and solve a linear programming model that proceeds by constructing spanning trees over the obtained mesh. Such validation leads to the evaluation of the peering strategies independently from the scheduling strategy that is used. The model has a good scalability and can be extended to reflect the view of a P2P designer or to find the most ISP-friendly strategy among different overlay construction strategies.

With respect to the content retrieval part, we combine the low scheduling delays of push scheduling with the resiliency and multi-sender ability of mesh overlays. We propose a new pure push scheduling strategy, *PurePush*, where we replace the pull mechanism by a probabilistic push: Parents of a node push a packet with a relaying probability to reduce redundancy.

Two variations of *PurePush* are proposed and compared with respect to playback delay and redundancy/overhead traffic to a typical Push-Pull algorithm. *PurePush* significantly improves the playback delay experienced by peers in the situation where there is packet loss.

Acknowledgments

It is with immense gratitude that I acknowledge the support and help of my supervisors Brigitte Jaumard and Brigitte Kerhervé. Throughout the preceding years, they showed a lot of patience, motivation and courage for embarking with me in a new topic for all of us. I also want to thank them for encouraging me and having confidence in my work.

I am also grateful to the members of the PhD supervisory committee for their efforts in evaluating the present thesis.

This thesis would not have been completed if it was not with the support of many people: My lab mates (Samir, Thomas, Jad, Istiaque, T. Murillo, ...), the IT service of Concordia University, specially people managing the CIRRUS cluster and, last and not least, my friends specially Adel Hejeij, Arbi, Zaki, May, Walid, Abdelghani, Adel Serhani and Ikbal.

I am infinitely grateful to my parents and to my wife who always supported me, pushed me forward and did her best to let me work in the best conditions.

Contents

List of Figures

List of Tables

ACRONYMS

ALM Application Layer (or Level) Multicast.

CDN Content Distribution (or Delivery) Network.

DAG Directed Acyclic Graph.

FTTH Fiber To The Home.

ISP Internet Service Provider.

LAN Local Area Network.

MDC Multiple Description Coding.

NAK Negative Acknowledgment.

P2P Peer-to-Peer.

QoS Quality of Service.

RTT Round Trip Time.

TFRC TCP Friendly Rate Control.

VoD Video on Demand.

VoIP Voice over Internet Protocol.

xDSL x Digital Subscriber Line.

Part I

Introduction and Problem Statement

Chapter 1

Introduction

With the evolution and the widespread availability of broadband Internet access through cable modem, xDSL, LAN and FTTx (Fiber To The Home, Fiber To the Node, etc.) Internet users have seen their download bandwidth significantly increased. This opened up the door to multimedia services including, e.g., online gaming, Video on Demand (VoD) and Audio/Video live streaming which typically require sending the content from one source to many users.

Although, the best way to achieve multicasting over Internet is to use IP Multicast, it is, unfortunately, not well deployed due to several constraints. Alternatives to IP Multicast include server-based architectures such as client-server architecture and Content Delivery Network (CDN). However, the server may become a bottleneck in the system because of its capacity limitations in both the number of supported connections and the total upload bandwidth. Thus, these last solutions have a limited scalability and are costly to deploy.

An alternative to these infrastructure-based multicasting architectures comes from P2P networks where heterogeneous end-clients (peers) form, in a distributed or centralized manner, an overlay network and cooperate to accomplish some tasks. The P2P paradigm was popularized starting from 1999 through Napster [NAP06], an MP3

file sharing application. A P2P architecture is potentially highly scalable and flexible because it relies on peer resources instead of network infrastructure. P2P applications include file sharing, distributed computing and streaming.

Although P2P solutions are very attractive for video streaming, there are many technical issues that limit its efficiency. These issues are still challenging research problems and include:

1. *Churn Resilience* [BLBS03, RO03, SR06, WL05b, LN06]: Typically, any peer may join or leave (through quit decision or failure) a P2P overlay at any time. Moreover, a P2P system must be able to handle a large number of simultaneous join or leave requests (flash crowds), for example at the beginning of an event and at its end;

2. *Peer heterogeneity* [PWCS02, LLC09]: This is the consequence of the diversity of access networks. Peers have different nominal upload and download capacities that may vary significantly;

3. *Efficient and optimized overlay construction* [PWCS02, THD03, CDK+03, KRAV03, PKT+05, RLC08]: A P2P system must organize peers in an overlay where communication can be conducted efficiently and in a scalable manner. An overlay can be optimized towards maximizing throughput, minimizing delays, maximizing resiliency to high churn rates, etc. This implies the use of strategies to select sending peers. The more an overlay reflects the underlying physical network, the more it is efficient;

4. *Content retrieval mechanisms* [ZLLY05, ZLZY05, MR06, Liu07, BMM+08]: Once the overlay is constructed, an efficient mechanism must be provided to allow peers to get the content with optimized quality and delays. Such mechanisms must be able to recover quickly from data loss and to minimize received

data redundancy. This may imply content advertising between some peers;

5. *Suitable video coding scheme* [PWCS02, SNG05, WL05b, WL07]: To improve P2P streaming performance, a coding that is able to cope with hard network conditions must be used. It must be resilient to errors and be able to handle packet losses;

6. *Inter-ISP traffic* [AFS07, CB08, XYK+08]: P2P networks are generating a very high volume of traffic and thus, they are stressing ISP networks specially by increasing the traffic volume among ISPs. ISPs have a hostile attitude by throttling P2P traffic and limiting the free transfer volume of their consumer clients. New proposals are investigating a cooperation between ISP's and P2P platforms to construct ISP-friendly overlay networks based on the exchange of information about the network state and also by adding ISP owned super peers.

Despite these challenges, we think that, P2P networks, already gaining in popularity, are becoming one of the most efficient solutions for multimedia delivery over Internet.

In the present thesis, we are interested in live video streaming using a P2P network. More specifically, we will focus on the situation where a single source is streaming live content to end clients. Compared to file sharing or to Video-on-Demand, live streaming has additional constraints like the need to minimize source-to-end latency[1], the non availability of future video segments and the timely delivery of content.

Although source-to-end delay is a critical point in live streaming, specially for applications such as video monitoring and video conferencing, existing works do not propose a complete approach for delay minimization taking instead some intuitive decisions or focusing mainly on either the overlay construction or the content retrieval mechanisms. Many delay related issues remain to be addressed, e.g., those dealing with network delays, with overlay performance, with delay definition/estimation and

[1] the terms latency and delay will be used interchangeably in this document

with reduction, among a peer supplying nodes, of content discrepancy that causes higher latencies and playback interruptions.

Moreover, delay minimization alone is not enough to guarantee the quality of the video streaming because it has many correlations with other criteria like maximizing resiliency or throughput. In this research proposal, we describe our approach for delay minimization at each peer in the streaming session. This objective will be achieved by both an overlay construction protocol and a content retrieval mechanism.

Overlay construction (peering strategies) has been till now the object of many research works, e.g., [PWCS02, THD03, CDK$^+$03, RS04, LN06, RLC08]. Actually the problem is quite rich and important. An overlay with poor performance will certainly result in a vulnerable or low quality system. Challenges in overlay construction include peer membership management, overlay topologies, criteria-based overlay construction, monitoring of the connectivity quality with neighbors, etc. In addition, most peering strategies rely on local information only such as peer characteristics or end-to-end information. We aim at identifying new information, mainly about the state of a node in the overlay. Such information should help in constructing an overlay network that reduces the delays experienced by peers and would serve as a basis for the peering strategies we propose.

Regarding content retrieval, despite several works, recent mechanisms (based on a pull or a push-pull approach) require content reconciliation to locate missing data and their explicit request from the receiver. This introduces delay and adds complexity to the receiver side (it is responsible to schedule and manage data reception from sending peers). Without advertising, the receiver will face the problem of data redundancy which degrades the performance of the system.

Although some push scheduling strategies exist, they rely on a heavy process [BLPL$^+$08, WL07, BMM$^+$08] and consequently they are not practical enough to be

implemented in real systems, specially mesh-based ones. In addition, some of them still operate by content advertising to determine needed data or to reduce redundancy. Therefore, we aim at proposing a pure push scheduling strategy that operates with almost no scheduling delays and without content advertising. We have to keep the process as simple as possible while minimizing the risk of redundancy and overhead. A special mechanism is then needed notably at startup and to recover lost content.

The rest of the document is organized as follows. Chapter 2 gives an overview of P2P streaming solutions based on the description of overlay topologies, data retrieval mechanisms and performance metrics. Chapter 3 provides a detailed statement of the research work. Then, in Chapter 4, we provide a literature review and illustrate how existing works fall out short of the goal regarding the objectives of the proposed research.

The first contribution of the thesis lies in Chapter 5 where we investigate the impact of the characteristics of the streaming source and the participating peers on the P2P system performance and more specifically on playback delays experienced by peers. Through the use of a linear programming model, we identify related trade offs that need to be considered in order to fully exploit the potential of a P2P streaming system.

The second contribution consists in proposing new peering strategies with different variations which have been studied in Chapters 6 and 7. In Chapter 6, we propose to revisit the peering strategies with a focus on playback delay minimization. Such strategies will benefit from the push-pull mechanism as the pull part is used mainly at the beginning of the session or to recover lost content. We believe that making the right decisions about node relationships will boost the performance of P2P systems. We focus on peering strategy variations where a peer selects the minimum number of parents that provide it with the full streaming rate. Results show that one of

the proposed strategies outperforms significantly the existing ones with respect to playback delays experienced by participating nodes.

With resiliency in mind, Chapter 7 reconsiders new variations of the strategies being compared in Chapter 6: We impose a fixed number of parents to each node. The two strategies that we propose lead to the lowest playback delays: They lead to overlays with short paths to the source.

The third contribution deals with the validation of peering strategies: Chapter 8 validates the results obtained in Chapter 7. In addition it lays the first stone toward evaluating peering strategies independently of simulations/experimentations, implementations and assumptions. We propose a Mixed Integer Linear Programming (MILP) model to evaluate peering strategies independently of the scheduling mechanism. Besides being scalable and flexible, it may be used as pre-deployment analysis of the impact of design decisions.

The fourth contribution is presented in Chapter 9 which deals with scheduling strategies and proposes a new solution where we combine the low scheduling delays of push scheduling with the resiliency and multi-parent ability of mesh overlays. We introduce a pure push scheduling strategy, *PurePush*, where we replace the pull mechanism by a probabilistic push: Parents of a node push a packet with a relaying probability to reduce redundancy. Through simulations, we show that *PurePush* outperforms, with respect to playback delays, the push-pull strategy in the situation where we consider packet loss.

Finally, Chapter 10 presents the conclusion of the PhD thesis.

Chapter 2

P2P Streaming Overview

In this chapter we give a general view of P2P streaming systems. We start by the motivation of adopting a P2P architecture for video streaming instead of other existing solutions. Then in Section 2.2, we present different overlay topologies and see how they impact the streaming performance. In Section 2.3, we discuss mechanisms used to disseminate the content to all the peers. Section 2.4 describes mechanisms for overlay maintenance. Finally, in Section 2.5, we describe performance metrics at both the application level and the overlay level.

2.1 Why P2P Streaming?

2.1.1 Existing Streaming Architectures

IP Multicast. In the late 80's, it was argued that multicast should be implemented in the network layer instead of the application layer [Dee95] because the performance benefits of implementing multicast at the IP layer outweigh the cost of additional complexity [CRZ00]. IP multicast works through building a multicast tree at the IP layer. In such a tree, the nodes are routers with high speed transfer and high

reliability but with limited processing and extensibility capacities. IP multicast is efficient as it neither introduces delays nor duplicates packets on the same physical link. This is made possible by the fact that routers are being responsible for content replication. Although most routers support IP Multicast, it is not widely deployed because of the following reasons [CRZ00]:

- Scalability as routers must maintain per multi-cast group state (which also violates the stateless architecture of the Internet);
- High level services such as reliability, security, error control and congestion control are difficult to implement in the multicast case while they are well mastered in the unicast case;
- Need for infrastructure change.

In addition to the lack of incentives to handle multicast traffic, IP multicast tree construction is not flexible. For example, it does not take into account criteria such as bandwidth while constructing the distribution tree.

Client-Server Architecture. This is the classic architecture where each end-client has a unicast connection to the source server. The same data is sent on each of the upload links which can be seen as a loss of resources. The main problem with this architecture is its lack of scalability. Indeed, the streaming server will be a bottleneck for the system because it has a limited number of simultaneous connections (typically between hundreds and few thousands) and a limited upload bandwidth.

Content Distribution Network (CDN). CDN is seen as a content delivery infrastructure-based network. The main idea is to strategically deploy several servers in different geographic locations in the network. Content is replicated at the servers. An end-user request is transparently redirected to the best server (that optimizes

given QoS criteria). CDN is a costly solution with limited scalability: it requires collaboration among several ISPs, must avoid bottlenecks at the server side and needs an optimized decision on where to place those servers.

2.1.2 P2P Architecture for Video Streaming

Application Layer (or Level) Multicast. Application Layer Multicast (ALM) is a specific example of P2P networks where the multicast functionality is managed by the application layer instead of the network layer. The multicast is achieved, in a distributed manner, by using several unicast end to end connections among peers. This involves content replication and retrieval, membership management, peer group communication, etc. The idea of ALM was revived in [CRZ00] and, from then on, was used in Internet conferencing, video streaming, on-line multi-player gaming [RM05], VoIP [GWYS05], etc. In [CRZ00], the authors differentiate between two ALM types:

- Proxy-based (or infrastructure-based) ALM where some application-level proxies are located throughout the Internet to assist end-client nodes to form a high quality overlay;
- P2P architecture where end clients are organized in an overlay network. Each end-client acts simultaneously as a server and as a client.

The P2P paradigm was made popular by file sharing applications such as Napster, Kazaa and BitTorrent. A P2P architecture does not require Internet infrastructure modifications. These clients share their resources (bandwidth, storage, CPU,etc). Moreover, they are software extensible and participate actively in content distribution. The P2P architecture is potentially highly scalable as the resources grow with the growing number of joining peers without any administration cost or network infrastructure support.

In a P2P network, clients are organized in an overlay network. A link between two nodes in the overlay maps to one or more unicast path(s) in the physical network (Internet), see Figure 2.1. Thus, the overlay network is independent of the physical infrastructure.

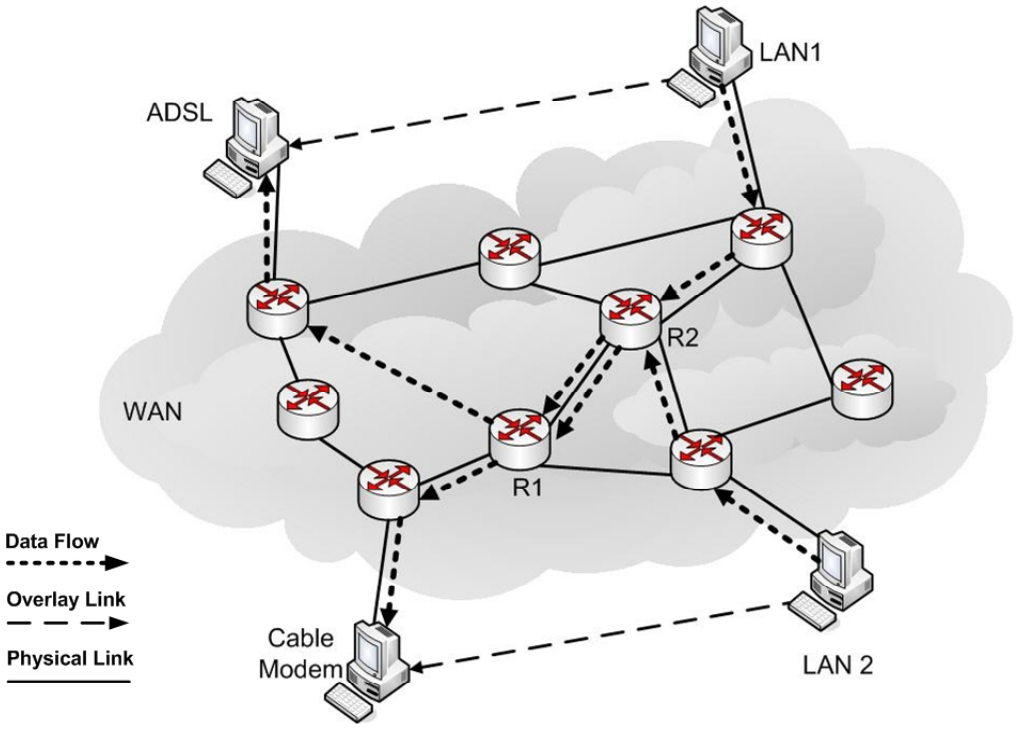


Figure 2.1: Overlay Network Example

P2P Streaming. P2P video streaming breaks with the classic client-server view and provides an architecture where end clients are able to get the same content from one or many other clients (possibly not including the source server). Each end client is participating with its own resources. The P2P solution is highly scalable. Moreover, it eliminates bandwidth bottleneck at the streaming source and reduces the cost of deployment. P2P paradigm in video streaming leads to many streaming configurations, i.e., one source to many receivers [KRAV03, ZLZY05], many sources to many receivers [CRZ00], many sources to one receiver [MAM06]. These configurations

are all compatible with live or video on-demand (VoD) streaming. P2P streaming systems, as any streaming system, are delay sensitive and bandwidth demanding. The most important objectives are to guarantee playback continuity and video quality.

Unfortunately, along with traditional streaming problems such as network congestion and bandwidth fluctuation, P2P streaming systems must cope with common issues of overlay networks which include:

- High stress on the network where multiple overlay paths may go through the same physical link, e.g., link R1-R2 in Figure 2.1. Moreover the same data may go through the same link several times;
- Unpredictability of peers (churn): dynamic change in overlay membership/topology caused by joining, leaving and failing peers;
- Heterogeneous and asymmetric (download vs upload) bandwidth of connections among participating peers;
- Fluctuations in upload bandwidths and in end-to-end delays among peers;
- Inter-client communication which introduces overhead. This communication will allow overlay maintenance and data retrieval (such that each client knows what data is needed and from where to get it). Overhead has to be small to keep P2P streaming efficient.

P2P live streaming adds some other constraints such as the playback delay of frames or segments and the non availability of future segments. This is in contrast with file sharing applications where all blocks of data already exist. Video on-demand has an additional important issue which is the asynchronous availability of content because peers will start sessions at different times and they must get all the content from the beginning. This is not the case in live streaming as a joining peer will only get the

content that is disseminated at that time.

Sripanidkulchai *et al.* [SGMZ04] have demonstrated the feasibility of large scale live streaming in overlay networks based on three key requirements: the availability of resources to construct the overlay, the maintenance of connectivity and stability in the overlay and the efficiency.

2.2 Overlay topologies

P2P streaming protocols organize peers in an overlay network and specify mechanisms for content delivery. Some of them differentiate [KRAV03, CRZ00] between the data plan (overlay used to disseminate data packets) and the control plan (overlay to disseminate control packets). In the following, we describe the overlay topologies for data dissemination. These topologies impact both the overlay performance and the streaming quality. The most used ones are tree, multi-trees and mesh topologies.

2.2.1 Tree

Principle. All nodes are organized in one unique tree rooted at the streaming source. Some nodes will have direct connections to the source and some will have other peers as parents. A parent must have the sufficient bandwidth to provide each of its children with the video stream. Usually, the source is uploading the same data (or packets) on each of its links. The first studies on overlay networks used the tree topology trying to emulate the multicast tree of IP, e.g.,[CRZ00].

Advantages & Drawbacks. An overlay tree is highly scalable and efficient in terms of physical link stress and control overhead and end-to-end latency [BB04] (depending on the height of the tree). Also, routing decisions are simple and predictable. However, the tree structure has many drawbacks:

- Any loss high in the tree will cause a loss down in the tree. Thus tree-based systems are highly sensitive to network congestion. In addition, the failure or leaving of a parent can cause the interruption of the playback;
- The incoming bandwidth of a node is limited by the upload bandwidth of its single parent.
- The transmission rate must be lower than upload bandwidth of the nodes in order to be able to construct the tree;
- Inability to handle the heterogeneous and the asymmetric bandwidths of peers;
- Bandwidth is wasted as each outgoing link of a peer is carrying almost the same information;
- Maintaining a high bandwidth overlay tree requires continuous probing for a possible new parent with higher available bandwidth than the current one. This is necessary in order to react to the dynamic evolution of the overlay tree but implies higher overhead [KRAV03];
- The tree must be balanced in order to minimize the number of children of a failing/leaving node but this increases the number of free-riders.

Free riders are nodes for which available outgoing bandwidths are less than the streaming rate or that are not participating with their upload bandwidth. In a tree, leaf nodes are also free-riders because they do not have children so they only get the stream and do not upload it to any other peer. Thus in a tree, where each link carries all the stream, these nodes can not have children.

2.2.2 Multi-trees

Principle. Instead of a single tree, the overlay network is organized into several trees rooted at the streaming source. Each node participates at every tree and is likely to have different parents and children. To improve efficiency and stability, some other constraints may be imposed such as the fact that one node is an intermediate one in exactly one tree [CDK+03]. In the other trees, it must be a leaf node. Thus, only one tree will be affected by the departure/failure of this node. If the paths from a node to the source (one in each tree) are not disjoint, then the failure of a common node will cause the first node to be disconnected from more than one tree.
Each tree will carry a substream of the video stream. All substreams are usually disjoint or with little redundancy. This is achieved by using encoding techniques such as layered encoding [HSLG99] or Multiple Description Coding (MDC) [Goy01]. The advantage of using MDC instead of layered encoding is that each part of the stream can be understood by the receiver while in layer encoding the client must have a particular layer called *base layer* to be able to benefit from the higher layers.

Advantages & Drawbacks. A multi-tree overlay is more resilient because it is unlikely that one node will see all of its parents fail. The most common case is when one parent leaves or fails so the node will be affected in probably only one tree. As a consequence, it may encounter a quality loss but no playback interruption.
Multi-trees with MDC also allow the increase of available bandwidth resources in the overlay comparing to the single tree case because some free-riders in single trees can participate with their low upload bandwidth to some of the trees [SGMZ04]. Thus, multi-trees are usually more efficient than a single tree. However, there is more overhead because each tree has to be maintained. In addition, each node will

encounter more ancestor changes compared to a single tree. Other issues with multi-trees includes the difficulty to construct different trees that optimize a given objective. In addition, connections between trees may depend on a single bottleneck [RS04].

2.2.3 Clustering

Principle. Clustering is used with a high number of peers in order to make the management process easier. Each cluster has a Membership Server (MS) or head cluster. A joining node gets a list of head clusters and selects one server to contact. It gets a fresh list of peers belonging mainly to the cluster managed by the contacted MS. A new MS is chosen by the old MS or by the Rendez-Vous Point (RP) who provides a full or partial list of nodes participating in the overlay. Keep-alive message exchanges are necessary among:

- RP and MSs to recover from MSs departure or failure;
- MSs: some of them will be included in the keep-alive message sent to members;
- MS and its cluster members.

Clustering policies may be [SGMZ04] random, based on delay or based on geographic proximity. A cluster must have a maximum size (to avoid overloading the MSs) and enough resources (to use local hosts as download sources) [SGMZ04]. Clustering does not impact the stability of the overlay as in the case of a tree but improves efficiency (with delay-based clustering) [SGMZ04]. Clustering is mainly used in tree-based systems such as ZIG-ZAG [THD03], Nemo [BB04] and AnySee [LJL+06]. One of its drawbacks is that it may impose high load on some particular nodes.

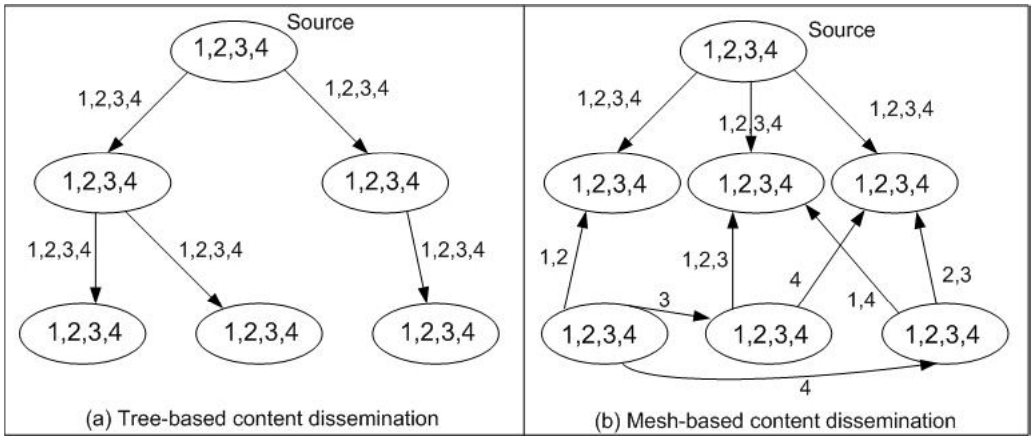


Figure 2.2: Tree vs Mesh based content dissemination

2.2.4 Mesh

Principle. A mesh topology is an example of an unstructured overlay where any link between two peers is virtually possible. P2P mesh overlays have been widely used in file sharing applications and more specifically with the BitTorrent protocol. In BitTorrent [BIT11], a file is divided into several blocks. Peers that possess the whole file are called seeds while peers having some part of the file are called leechers. These peers participate in a swarming activity where, each leecher downloads blocks of data from different neighbors that are obtained from a tracker server. The downloaded blocks are not necessarily contiguous (the order is not important). Such a technique is not feasible in video streaming because there is a timeliness requirement for playing downloaded content.

In a mesh, a node may receive data from more than one node and simultaneously send data to one or many nodes. Such node then increases its data reception rate but has to guarantee that it is not receiving the same data from many parents, i.e., the data sets on each of the node's downlinks must be disjoint. To illustrate the multi-sender characteristic in a mesh overlay compared to a tree overlay, Figure 2.2.3 gives an example of content distribution of four video segments (1-4) in both a tree

overlay and a mesh overlay.

In some cases, the node is responsible for selecting some of its parents [KRAV03]. These parents or neighbors must be able (bandwidth and content wise) to provide missing data to the requesting node. There are two types of mesh topologies: unstructured mesh (represented by an undirected graph) and Directed Acyclic Graph (DAG).

A mesh overlay is inherently resilient to churn because there is no need to build and maintain a specific structure such as a tree or a DAG. Both peer membership management and content distribution may follow gossip/epidemic algorithms [EGmKM04]. Typically, each peer sends a message to a random subset of neighbors until the message is forwarded to all peers. The random selection of destination nodes achieves the resiliency to churn and packet losses. With respect to video streaming, gossip algorithms may lead to higher delivery times and received data redundancy.

For instance a randomly constructed mesh overlay is usually associated with a gossip protocol that enables decentralized distributed communications (membership management and content advertising) among peers [KRAV03, ZLZY05, ZLLY05]. A node sends a message to a random set of other nodes participating in the streaming session. Each of these peers sends the same message to another random set.

On the contrary, DAG [LN06, Ooi04] imposes some properties on the mesh to improve connectivity. For example, in [LN06], each peer has at least k parents so the vertex connectivity is equal to k which means that every simultaneous failure of $k-1$ nodes does not cause the network to be partitioned.

Advantages & Drawbacks. A mesh architecture is more robust and resilient to high churn rates. It is also able to achieve high throughput. Indeed, downloading data from many nodes improves both throughput (not limited to available upload bandwidth of the parent) and reliability (reduces the impact of single node failure).

Overhead in mesh overlays depends on the maximum number of neighbors per node. It is usually more important than in tree or multi-tree topologies. Indeed, each node must continuously check whether its neighbors are still alive. It is also responsible for identifying nodes that have the content it is looking for.

2.3 Content Retrieval Mechanisms

The two main approaches for content retrieval are the push and pull approaches.

Push Approach. In a push mechanism a node only forwards data to others following some rules. The receiver does not request any data. In this type of mechanism, data loss can be hardly recovered by the receiver. The push mechanism is simple in a one parent context (for example: trees) but is hard to deploy in a multi-parent context because of two main issues: data redundancy and data loss. In addition, when a node is missing a packet, it is difficult to state whether the packet is lost (so the receiving node has to request it) or will be pushed by a parent and thus, the receiving node has to wait for it. System examples mostly include tree and multi-tree based systems.

Retrieval of Content by the Receiver. This is also called pull mechanism. It may be based on data chunks, i.e., temporal division of the video stream or on sub-streams, i.e., spatial division of the video stream.

In receiver-driven systems, a node has to be aware of the existing data in some of the overlay nodes (neighbors for example). Then, it explicitly requests missing data from a subset of these nodes. This is referred as *content reconciliation* [BCMR04].

Many techniques have been proposed [BCMR04] but they are relatively delay and resource intensive. They may need larger buffers at both sender and receiver peers.

When using a pull mechanism, lost data is easily recovered. However, it increases delay because of the three steps model used: advertise content, request and send. The pull method can be seen as complementary to the push approach and they can be combined [ZLZY05].

Pull-based systems suffer from an efficiency vs. low delay tradeoff. Indeed, increasing the buffer map advertising period among neighbors will reduce the overhead but will increase the delay. On the opposite, reducing the advertising period will increase the overhead even if it helps in reducing playback delays.

2.4 Membership Management

Membership management includes operations for a peer to join and leave the overlay as well as making sure that some peers are still participating in the overlay. Communication can be centralized in which case a peer or a server must have a global state information of the overlay which is not a scalable solution [CRZ00]. It can also be distributed in which case a peer maintains information about a subset of the overlay only.

Join. The join operation is achieved using a Rendez-vous Point (RP) that provides a full or partial list of nodes participating in the overlay. The RP is generally different from the source server and is known to all clients. The received list can be random as in [SNG05]. The new peer then selects possible parents and contacts them.
Parents can be selected randomly, based on the Round Trip Time (RTT) or more generally the end-to-end delay, the available throughput or the geographical proximity. Bandwidth based criteria is not scalable because it requires periodic probing while delay based criteria does not guarantee enough bandwidth. Indeed, it is possible that a node has low delay but also low bandwidth while another node have longer

delay and higher bandwidth. More complex selection criteria may be used. In the experiments of [SGMZ04], a node tries to predict how long each possible parent will stay in the overlay using some heuristics and then selects the parent with the highest lifetime first. The diversity of parents is a good step towards overlay resiliency and robustness.

Departure. A departure is graceful if before disconnecting, a node informs its parents and its children so they can reorganize themselves. For instance, in a tree based system, each descendant tries to connect independently to the tree. Non free-riders nodes are prioritized.

Failure. It is an ungraceful departure due to node failure (hardware or software) or even network failure. Parents and children are not aware of it and must detect this failure. This is done through maintenance.

Maintenance. Maintenance is achieved through periodic exchange of hello like messages. Each node has a life time for each of its neighbors. If it does not receive anything after some time from its neighbor, it assumes that it has failed. Each node maintains a list of spare parents to contact if some of the current parents leave or fail. Maintenance may include dynamic improvement of the overlay with node probing for other existing members to find better parents [CRZ00, RLC08].

2.5 QoS

We can consider that in application layer multicast ALM there are two types of QoS. The first is related to end-client application performance and the second is related to the overlay performance. It is obvious that the overlay performance has some impact on the application performance.

2.5.1 Overlay Efficiency

Sripanidkulchai *et al.* [SGMZ04] define the overlay efficiency as the degree at which the overlay reflects the underlying IP network. Based on that, overlay efficiency may include the connectivity performance of the overlay and the cost in terms of overhead.

Some other important metrics include physical link stress and the rate of useless packets [BB04]: duplicated packets or packets that arrived out of the delivery window.

Quality of Connectivity. We can define the quality of the connectivity as the ability of the nodes in the overlay to send and to receive data at the right time for their use. Birrer *et al.* [BB04], define connectivity as the percentage of nodes that received at least one packet in 10 sec. Sripanidkulchai *et al.* [SGMZ04], evaluate connectivity based on two metrics: *Mean interval between two ancestor changes* and *Number of descendants of a departing node*. The first one is an indication of the connectivity performance seen by a node. The second one is an indication of the system connectivity. The greater this number is, the poorer is the connectivity because a departure will affect many descendants. Requirements for a good connectivity include stability and scalability.

Control Traffic. Control traffic includes communication among peers to exchange video packets information or messages used in overlay maintenance. A good multicast protocol must have low overhead. When using a gossip protocol, having a limited number of neighbors makes the associated overhead independent from the overlay size. Thus, it adds scalability to the system.

2.5.2 Streaming Performance

Streaming performance metrics include:

- Bandwidth/throughput;
- Latency, i.e., source to end delay [BB04], time between sampling time at the server and playback time at the receiver. It may also be estimated by the number of hops;
- Playback deadline, i.e., playback time of a specific data after which it is useless;
- Playback continuity, .e.g, continuity index, defined in [ZLLY05], which is the ratio of segments that arrive before or on playback deadlines over the total number of segments. It does not significantly improve with a rising number of partners over a given threshold [ZLLY05];
- Start-up delay, i.e., time before the playback begins;
- Delivery ratio, i.e., ratio of in-time delivered packets over total generated packets [BB04, ZLZY05].

Although these metrics are known in the classic client-server streaming architecture, some of them become more critical in the P2P streaming system such as all delay related metrics, because they are no more related to the physical network only but also to the overlay performance.

Chapter 3

Problem Statement and Contributions

In the following, we describe in detail the research conducted within this PhD thesis. In Section 3.1, we formulate the global problem that we are tackling and which deals with a delay-centric P2P live video streaming and give the set of assumptions and limitations. We have defined two main objectives: Proposing a new peering strategy and a new content retrieval mechanism that are focusing on minimizing playback delays experienced by peers. For each objective, we discuss its motivations, the proposed research and the contributions in Section 3.2 and in Section 3.3 respectively. Finally in Section 3.4 we summarize the research outcomes.

3.1 Main Problem Formulation

3.1.1 Global Problem

Several studies have been conducted in order to design and deploy P2P systems in the context of live video streaming over Internet, investigating efficient and optimized overlay construction [PWCS02, CDK+03, KRAV03, RLC08], content retrieval

mechanisms [ATS04, ZLLY05, ZLZY05, MR06], etc. Most of them mainly focus on maximizing throughput or resiliency while trying to keep the playback delay (source-to-end streaming latency) bounded, with this latter delay often estimated by the number of hops, at the overlay level, from the source to the end node.

In addition, most existing systems do not consider situations where delays are critical to the performance such as interactive applications. Indeed, they focus on Radio/TV broadcasting, on-demand videos, where delay related requirements are important but relaxed toward low cost deployment and video quality.

P2P live streaming is not exclusive to such applications. For instance, it may be a tool to broadcast a live conference or live e-learning session where users want to interact through asking questions using text messages, voice or video. Other applications may include live video on-line auction, video surveillance and military communication. The streaming delay is even more critical in such a case.

However, delay has not been given the importance it deserves. Indeed, delay from the source is a crucial point in live streaming, and thus, from our perspective, minimizing or bounding it, is a key issue.

In the present study, we are investigating the problem of providing a live video streaming session using a unique streaming source. We propose to examine the objective of minimizing the streaming delay experienced by every node in the P2P overlay network. In addition, we consider an overlay with a mesh topology as mesh overlay networks are believed to be suitable for P2P streaming and inherently resilient to failures, see, e.g., [YCM+04].

The possible actions in order to achieve the proposed objective are limited to the phases of overlay construction/maintenance and of content retrieval. Thus, we will deal with two main contributions: an overlay construction protocol and a content retrieval mechanism.

3.1.2 Assumptions and Limitations

In this section, we describe the main assumptions we make:

1. *Sufficient download bandwidth*: Every peer in the session has enough download bandwidth to accommodate the streaming rate. This is a reasonable assumption because Internet access is continuously evolving with the deployment of recent technologies like ADSL2+ and FTTH access networks. So, in the near future download bandwidth will not be an issue;

2. *Non awareness of underlying physical network*: We assume that nodes do not have any knowledge about the physical underlying network. Although it may result in some problems related to shared paths for streaming (shared bottleneck, unbalanced load, link stress), it is a realistic assumption. Otherwise, nodes must cooperate to discover physical network topology and to assign paths to data flows. This adds complexity to the receiver side and we will not address those issues;

3. *Limited upload bandwidth*: The upload bandwidth of peers is the most important resource. A bad utilization will result in a poor performance of the global session. The uploading is still an issue even with new access technologies. In our work, we will consider that the upload bandwidth of a peer may be lower than the streaming rate;

4. *Network Address Translation (NAT)/Firewalls*: The impact of firewalls or NAT on node connectivity and effective upload bandwidth is not being considered in the present work. It is left for future work.

5. *Constant bit rate at the source*: We consider the situation with a constant bit rate because it gives more control on the scheduling strategies and on the

mechanisms that may be used to recover from data loss. Indeed, with a constant bit rate it is easy to identify which data has been received and which data has been lost.

6. *Same quality for all*: In our proposed research, scalable video was not considered. Thus each participating peer aims at receiving the full video stream.

7. *Unfair upload bandwidth exploitation*: It is up to a participating user to limit the upload bandwidth that he wants to offer to the streaming session. Our solutions will assume that these amounts of bandwidth are available.

8. *Jitter*: Jitter in packet delivery is not being considered. Studying the impact of such phenomena is left for future work.

In the present work, we do not tackle practical issues such as security, digital right management, legal aspects, etc. For video coding, we refer to works providing encoding resilient to errors and packet loss. Also, we will not target P2P streaming in a mobile environment.

3.2 Delay-centric Overlay Construction

3.2.1 Motivations

In many existing works, the common practice is to propose a P2P streaming protocol based on a multi-sender receiver-driven approach using a mesh topology. Usually, the mesh overlay uses a gossip protocol for membership management. A joining node will get a random initial set of peers known to be in the streaming session and then will select some parents randomly or using a criterion. However, many uncontrolled and unwanted situations may occur causing degradation in the streaming performance. For example, there is no guarantee of efficient utilization of the upload bandwidth of

a parent as it is the receiver who decides what data the parent has to send.

The authors of [LN06] point out some connectivity (risk of mesh partitioning) and loop problems in randomized mesh topologies. It is thus clear that a topology must have some properties to avoid such problems. Unfortunately, the more constraints and properties an overlay has, the more it is sensitive to churn. The challenge is to impose properties that may be recovered easily and quickly.

Although delay is usually expressed in the number of hops between a peer and the source (e.g., [MR06, SLL06]), it is not enough to precisely describe the effective delay. Indeed, a peer at a low level in the mesh (low distance from the source in number of hops) does not necessarily have lower delay than another peer in higher level. For example, we may prefer high download bandwidth peers to be close to the source but this does not mean that they have the lowest end-to-end delay to the source. More relevant information is needed to reflect more realistically the delay state of a node.

3.2.2 Proposed Research

The objective of this step is to propose new peering strategies in order to construct a mesh overlay that focuses on delay minimization. We exploit the fact that, recently, several mesh-based P2P live streaming systems are adopting a push-pull mechanism instead of the classical pull mechanism. A push-pull mechanism is more efficient in terms of overhead and leads to much better playback delay performance because it eliminates the need of the three steps of pull content retrieval: Buffer map broadcast, data request and data sending. Thus, using the pull mechanism is not the best way to evaluate the performance of peering strategies especially the ones targeting playback delay minimization.

Since a push-pull mechanism typically leads to low scheduling delays, the content delivery delay is impacted mostly by the overlay path length. Thus, we propose to

revisit peering strategies in mesh overlays using the push-pull mechanism proposed in [ZLZY05, ZZSY07]. In addition, thanks to the fact that the push-pull strategy restores the importance of the overlay path lengths, the delay estimation by a number of hops is no more suitable and must be replaced by the delay observed on the overlay links.

This stage led to the proposal of new a peering strategy, *BestParents*, that is based on the selection of parents that offer low delay paths to the source (See Chapters 6 and 7).

Unfortunately, proposing a peering strategy that performs well through simulations or experimentations is not enough to compare them with existing ones. In fact simulation and implementation scenarios combined with design choices impact the performance of any strategy. Thus, despite the advances made by P2P streaming systems, they have failed to be in the foremost of content distribution solutions.

On the one hand, P2P deployment is suffering from the digital rights issue and from the hostility of ISPs who try to minimize their inter traffic. On the other hand, it is very difficult to guarantee the QoS performance such as playback delays and streaming quality as they usually rely on experimental evaluations.

As part of some initiatives for Future Internet, new ideas call for a collaboration (based on topology information and even hardware support) between P2P systems and ISPs [ABE+09]. To be more effective, we believe that they must be complemented by a performance evaluation methodology that is less prone to design decisions such as scheduling strategies and implementation choices. Thus, we will also propose a MILP model for evaluating peering strategies.

3.2.3 Contributions

Through working on the overlay construction part, we made the following contributions:

1. We studied the impact of the characteristics of peers and of the source node on the experienced delays by peers and the identification of tradeoffs related to those values, which led to the following publication:

    - Anis Ouali, Brigitte Jaumard, Gérard Hébuterne, Trade-offs in Peer Delay Minimization for Video Streaming in P2P Systems, Eighth International Workshop on Global and Peer-to-Peer Computing. In Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid, CCGRID'08, May19-22, 2008, Lyon France

2. We proposed new peering strategies that exploit information about the state of nodes in the overlay mainly their delays to the source to make wiser selection decisions, and

3. We compared those strategies with the state-of-the art ones using newer content retrieval mechanisms, mainly a push-pull mechanism instead of a pull mechanism.

    - Ouali, A., Kerhervé B., Jaumard, B., Toward New Peering Strategies for Push-Pull Based P2P Streaming Systems. Ultra Modern Telecommunications & Workshops, 2009. ICUMT '09. International Conference on , P2PNet09, October 12-14 2009, St.Petersburg, Russia.
    - Ouali, A., Kerhervé B., Jaumard, B., Revisiting Peering Strategies in Push-Pull Based P2P Streaming Systems, In proceedings the IEEE International Symposium on Multimedia ISM'09, December 14-16 2009, San

Diego,CA, USA. (Acceptance Rate 19.6%)

4. We validated the obtained results through proposing a MILP model. The model is easily scalable and customizable to reflect the view of a P2P streaming protocol designer or an ISP and thus it allows to evaluate a peering strategy without relying on simulations or specific implementations of the protocol:

   - Anis Ouali, Brigitte Jaumard, Brigitte Kerhervé, A MILP Model for the Validation of Peering Strategies, submitted for publication.

3.3 A New Content Retrieval Mechanism

3.3.1 Motivations

A content retrieval mechanism allows an end-client to receive data from other nodes using an already constructed overlay. Through existing works [HHB+03, BLBS03, RS04, ZLLY05, ZLZY05], it has been demonstrated that a multi-sender mechanism achieves better results than a single-sender one. In such a context, a receiver-driven approach was, for some time, the most suitable one as it allows the receiver to cope with two main challenges: Eliminating data redundancy and recovering from data loss.

However, a receiver-driven approach adds complexity to the receiver side because the receiver is responsible for scheduling the data sent by its parents/neighbors. In [ZLLY05] for example, the authors use buffer maps to advertise available content so that a node is able to request a given content from neighbors. These maps are periodically sent by each node to its neighbors. This adds overhead and load on peers. In addition, as already mentioned in Section 2.3, there is more delay because of the three steps of data retrieval (advertise-request-send). Minimizing the use of a pull

approach will result in minimized delays. Gridmedia [ZLZY05], illustrates the benefit of a combined push-pull mechanism on observed delay. A client registers itself with some neighbors that will systematically send data to him (push) and the pull is only used when there is missing data. This system is presented in more details in Section 4.3.

However, in Gridmedia, there is no guarantee on the pushing rate. In presence of a complex overlay, the push-pull strategy may trigger the pull part and thus increase the delivery delays and the overhead.

An efficient content retrieval mechanism will make efficient use of the upload bandwidth of peers and minimize the overhead. It has also important impact on minimizing delay and maximizing throughput or resiliency. Actually, we consider the content retrieval mechanism as the way to achieve efficient utilization of the constructed overlay network. Indeed, the overlay satisfies some quality requirements but it does not impose how it is used.

For example, a node may have parents with available upload bandwidth but if they don't have enough content to send, the bandwidth observed will be lower than expected: Content Bottleneck [MR06]. Moreover, when a receiver node assigns the most part of the content to only one of its parents, it will be sensitive to its departure or failure and then may experience significant degradation of quality. Hence, the system has a weak resiliency.

3.3.2 Proposed Research

The objective of this step is to propose a content retrieval mechanism in order to minimize the delays based on a realistic delay modeling.

Content retrieval is closely related to content granularity which may depend on the encoding used. When using layered encoding or MDC, the objective is to deliver

each content layer by layer. The content redundancy and data loss will be dealt at a layer level. However, rateless encoding, e.g., [BLMR98], deals with the delivery of a minimum number of different blocks of data. Traditional encoding deals with known blocks.

We set as a starting point the push-pull mechanism used in [ZZSY07]. We observe that the performance of the strategy depends on the complexity of the mesh overlay and may degrade when facing packet loss. In both cases, the pull mechanism is triggered. Thus, we propose to investigate and to propose a pure push scheduling strategy that eliminates completely the use of the pull mechanism.

Replacing the pull mechanism in a push-pull strategy will require finding a mechanism to get packets at startup and a mechanism to recover lost packets. In addition, at each step, we need to ensure that there is low redundancy and overhead.

To further satisfy the objective of delay minimization, we have to let every peer implicitly know from where to get the content. Thus, according to some rules a node knows what content it has to forward and which parents to choose to get a specific content.

We stick to mesh overlays because of their inherent resiliency to churn and their multi-parent ability. Push scheduling over mesh overlays is different from multi-trees. Indeed, we do not construct a multi-tree overlay. We construct a mesh overlay and then each substream will be distributed over a tree on top of the mesh.

3.3.3 Contributions

Through working on the content retrieval part we made the following contributions:

1. We proposed simpler scheduling mechanisms at both the receiver and sender side: A receiver is not responsible any more for coordinating the data to be sent from its parents while a sender does not need to advertise its buffer map any

more;

2. We designed a basic version of a pure push strategy that outperforms a typical push-pull mechanism with respect to playback delays of peers in the situation where there is packet loss:

    - Anis Ouali and Brigitte Kerhervé and Brigitte Jaumard, A Packet-loss Resilient Push Scheduling for Mesh Overlays, in Proceedings of IEEE Consumer Communications and Networking Conference, January 2011

3. We designed an advanced version of the pure push scheduling strategy that outperforms the basic version with respect to playback delays and where we succeeded at having very low redundancy both at startup and when recovering lost packets:

    - Anis Ouali and Brigitte Jaumard and Brigitte Kerhervé, Push Scheduling and Mesh Overlays: The Best of Both Worlds, submitted for publication.

3.4 Conclusion

In this chapter, we have proposed to achieve delay minimization at each peer in a live streaming session using a mesh overlay. We divide our work in two parts. The first one deals with overlay construction and maintenance while the second deals with a content retrieval mechanism. We have adopted a delay-centric approach for overlay construction through three main points:

- Using a more accurate delay definition that takes into account the network delay among nodes;

- Using new information describing the status of a node in the overlay such as its delay to the source;

- Determining the best tradeoffs between minimizing delay and other objectives such as resiliency, throughput and scalability.

The content retrieval mechanism is complementary to the overlay construction. It is based on two points:

- Eliminating the exchange of buffer maps in order to speed up the startup and the lost content recovery;
- Reducing the risk of redundancy and overhead.

Chapter 4

Literature Review

In this chapter, we present a literature review of the most representative P2P streaming systems. Section 4.1 describes possible categorizations of P2P streaming systems. In Section 4.2, we briefly describe the most relevant studies with respect to the problems we have proposed to study in Chapter 3, i.e., multi-sender overlay construction and content retrieval mechanisms. In Section 4.3, we present in detail one P2P live streaming system called GridMedia, to illustrate how current available systems work. We end this chapter by a critical analysis on the limitations and the drawbacks of the studies made so far.

4.1 P2P Streaming Systems Categorization

Based on the content retrieval mechanism Liu *et al.* [LYKM06] have divided P2P systems into two classes: Sender-driven and receiver-driven. In the first approach, the sender is responsible for coordinating related receiving peers and for stream distribution. In the second approach, the receiver is in charge of coordinating related sending peers and balancing load among them. The receiver can also use measurement and optimization techniques to help him decide which senders to choose, which

packets to receive from each receiver in order to, e.g., maximize throughput [RS04] or to minimize delays [WL05a]. The authors in [LYKM06] also propose another categorization based on the overlay topology: tree-based or treeless-based (DAG, Mesh). Although they are different, the two aforementioned categorizations are related. Indeed, usually tree-based systems use the push approach which is a sender-driven content retrieval while tree-less systems use a receiver-driven approach because of the multi-parent property.

4.2 Existing Work Review

P2P streaming systems and peering strategies have been studied in early works on P2P systems [SGMZ04, RS04] as well as recently in, e.g., [RLC08, HRV09, LLR09]. Evaluations of these proposed solutions were based on simulations or experimentations.

Several theoretical works also exist. They propose mathematical or stochastic models to study the performance of P2P streaming systems. For instance, the authors of [CCB07] model a P2P streaming system using stochastic graph theory. The video stream is divided into stripes. Each stripe will follow a diffusion tree over a mesh overlay. Arrival and departure of nodes follow an exponential distribution and peers belong to different classes according to their upload bandwidths. Results show, that the delay is influenced by the number of stripes. The higher is the number the higher is the delay. Also it is stated that increasing the redundancy among stripes leads to a better performance in face of a churn rate.

In [KR07], the authors propose a stochastic fluid model to analyze the characteristics of a P2P streaming system. They differentiate between super peers with high upload bandwidth and ordinary peers. They show that a minimum ratio of super peers to ordinary ones is necessary for the system to perform well. In addition larger

systems are believed to be more resilient to churn rates than small ones. Buffering is shown to improve the quality more than additional bandwidth provisioning.

However, optimizing the overlay construction is not a simple problem because of the lack of centralized information and the difficulty of dealing with the trade off *end-to-end propagation delay vs. upload bandwidth* of nodes: For instance, where do we have to place a node with a high upload bandwidth but with high end-to-end delays with the source or available parents/neighbors?

Therefore, several works estimate the delay as the number of hops. This way, they are able to come up with results such as a maximum theoretical bound for the playback delay [HRV09] ($O(\sqrt{\log n})$, where n is the number of nodes, or advanced peering strategies [LLR09]. However, such approximations ignore the end-to-end delay and therefore can not be considered as realistic estimations especially with peers located worldwide.

Alternatively, some works addressing playback delay minimization focused on content retrieval mechanisms notably through proposing push-pull mechanisms [ZZSY07, LMSW07] or improved scheduling strategies [Liu07, CXH08]. These solutions do not investigate how to construct an overlay that helps achieving such a goal. Typically, a joining node chooses neighbors randomly or based on the candidate basic characteristics such as proximity (RTT) and nominal upload bandwidth. Thus, they do not rely on information describing how well a node is doing in the overlay.

4.2.1 Studies with Main Focus on the Overlay Network

Many studies have addressed the issues related to the overlay construction, maintenance and performance in P2P streaming. Optimizing the overlay construction is not a simple problem because of the lack of centralized information and the randomness

that characterizes join/departure processes. Thus, most existing works use either basic or random techniques for parent selection and concentrate mainly on the topology being used: Tree, multi-trees, mesh, etc. Therefore, they do not rely on information describing how well a node is doing in the overlay.

For instance, the common strategies use random selection or a selection based on either the upload bandwidth of nodes or the RTT among nodes. The authors of [LN06] use the hop distance to the source to have an estimation of the delays experienced by peers. It is a coarse estimation, though, especially with nodes at different worldwide locations.

[CRZ00] is considered as one of the earliest work on application level multicast. The authors study the impact of adopting a multicast overlay on an application performance and how a good overlay can be constructed. A protocol for multi source multicast streaming called *Narada* is proposed. It is based on a mesh control plan with desirable performance properties. Over the mesh, and for each data source, a spanning multicast tree is constructed for data delivery. Membership maintenance is achieved by distance vector algorithm where each node has the list of all members in the session. Narada targets medium-size groups and does not scale to large size ones. ZIZ-ZAG [THD03] uses hierarchical clustering to minimize transmission delay and limits the node degree to bound node workloads. It also imposes some rules to logarithmically bound the height of the multicast tree and assigns the forwarding task of data messages and control messages to two different nodes in the cluster. Due to clustering some nodes will have the additional task of managing clusters and thus, the system will be vulnerable to their failure.

SplitStream [CDK+03] is a high bandwidth distribution system based on the existing Scribe overlay network [CDKR02] and Pastry overlay routing protocol [RD01]. It

aims at eliminating the problem of free-riders in a unique tree topology. In SplitStream, data is divided into stripes. One multicast tree per stripe is constructed and is responsible for disseminating the data. Each node has to join each tree but is an interior node in at most one tree. The last characteristic poses an important challenge but ensures fairness (a node never uploads more data than it downloads) and robustness (a leaving node affects only one tree).

CoopNet [PWCS02], differs from SplitStream by using a centralized approach, where the streaming source is the root of each streaming tree and is also responsible for information collection to construct and maintain the overlay. This centralized approach is efficient but obviously not scalable. In addition, the source server has a high load. CoopNet also proposes the use of MDC to cope with network heterogeneity with each layer transmitted over one tree.

PRO [RS04] is a framework for constructing P2P overlays designed for non-interactive streaming and that focuses on bandwidth maximization. PRO is based on a receiver-driven approach that uses an unstructured overlay mesh. Membership is managed through a gossip protocol to make a peer aware of a subset of participating nodes. Each peer maintains a local image containing a list of potentially good parents based on their access bandwidth (not the available one) and its distances to them. When needed, it selfishly and randomly selects parents that maximize its received bandwidth from its local image. Each parent controls congestion on its upload connections. This makes a receiver able to implicitly detect a change in the overlay structure or the occurrence of a shared bottleneck, i.e., a bottleneck at a physical link that is shared by the physical paths from each parent to the receiving node. PRO only deals with overlay construction and maintenance. The content retrieval is based on PALS [RO03] which allows the delivery of layer encoded content.

Promise [HHB+03] is a P2P non-live streaming system based on a P2P application layer service called CollectCast. CollectCast uses a receiver-driven multi-sender scheme and exploits topology and performance information of the underlying physical network to achieve better peer selection, sender and network monitoring, scheduling of data sending from parents (rate+data) and sender switching to respond to serious performance degradation. Promise is also built over a P2P substrate that provides membership management, peer connectivity and object look-up. It uses a topology-aware peer selection strategy (i.e, it tries to select peers for which the physical paths are not sharing some specific segments) to avoid correlation between performance degradation over the multiple paths that are used to deliver data to the receiver.

In [WL05a], the authors propose a distributed algorithm that computes optimal streaming rates assignment to the overlay links with the objective of minimizing source-to-peer latency. The algorithm makes use of a linear programming model that expresses the streaming as network flows to each peer in the session while minimizing the average source-to-end latency. Each peer assigns rate to its upstreaming nodes (parents). This average latency is computed based on delay cost and the streaming rate of each link in the overlay. Simulations have shown that the obtained solution outperforms a common heuristic that assigns rate to parents based on their upload bandwidth.

The authors, Wu and Li [WL05a] assume that the overlay network is already constructed. However, the delay cost of an overlay link is equal to 1. Such estimation will definitely not lead to the best possible delay performance. In addition, it is the source who triggers the computation of the rates for each peer in the session whenever a new peer joins or a peer becomes unable to recover from a data loss due to the departure or failure of a parent. This can be heavy to achieve even with a medium peer churn rate.

With respect to peers bandwidth management, the authors of [AKR+05] propose an optimization model for P2P streaming that minimizes the cost of used bandwidth to serve a receiving peer subject to the constraint of continuous playback even under one or multiple failures (using transmission redundancy). They use a pricing model that depends on the data to be transferred, the rate and the duration of the transfer. The cost can be different from one peer to another. Although it is an important point, in our work, we assume that we know the amount of available upload bandwidth for each peer and that there is no additional cost to select one peer instead of another.

DagStream [LN06] replaces the unstructured mesh overlay by a directed acyclic graph (DAG), with a specified minimum number of parents. This helps avoiding the occurrence of loops and partitioning of the mesh overlay which can result in higher delays or playback interruptions for the partitions that are disconnected from the source. DagStream tries to optimize the efficiency of network resource usage by using a multi-parent receiver-driven approach that emphasizes locality awareness over bandwidth maximization. Thus, a node primarily chooses nearby parents (with smaller delays to itself) but may also choose other parents to increase its receiving bandwidth. It uses the hop distance to the source to have an estimation of the delays experienced by peers. It is a coarse estimation, though, especially with nodes at worldwide locations. Additionally, a DAG will suffer from the same problems as a tree topology, i.e., some leaf nodes do not participate with their upload bandwidth. The selection process also prefers parents with lower levels from the source to minimize delays. Membership management is achieved by a service called *RandPeer* that maintains information about peers in the session and their QoS characteristics.

Chunkyspread [VYF06], presents a new tree based mechanism: Distribution trees are constructed using bloom filters (to avoid loops) over a randomly created graph. Each tree handles one slice of the original stream. Each peer specifies two load

parameters: The maximum load and the Target load. It will have a number of neighbors that is proportional to its target load. Two phase of tuning are taking place. In the first one, load tuning, the system tries to alleviate the load on overloaded parents and shift it to underloaded neighbors to achieve a node load within a specific interval. In the second one, latency tuning, nodes make parent switches to improve their latencies without violating the load interval form the first phase. Both phases require several messages among nodes and their neighbors in order to take the switch decision.

A more recent work [RLC08] takes a similar approach as us by trying to minimize delays based on network link delays of the mesh overlay. It proposes a distributed algorithm for overlay construction. When a joining node is looking for parents, it associates with each candidate a *Power* value depending on the candidate residual upload bandwidth, its distance to the source and the RTT. Then, a node requests the upload bandwidth it needs from the most powerful candidate parent. If it is less than the streaming rate, it looks for another parent. Thus a node may have one parent only. No mention is given on the retrieval mechanism being used (pull or push-pull).

Another problem is that although the authors claim to use mesh topology, their definition of the distance to the source as the longest path to the source suffers from the presence of cycles in the overlay. Such definition will lead the distance of some nodes to increase indefinitely unless they assume that, at the moment of joining, a node gets the full streaming rate from nodes already in the session which is not likely to be the case. In our work, we face the same issue and we propose a distance computing algorithm to deal with it.

As mentioned in [HRV09], the work in [RLC08] *may lead to low bandwidth utilization in the network* because the heuristic being used will assign a low selection probability to nodes at the edge of the mesh network. Thus, such nodes, will not

contribute significantly with their resources. This observation has been confirmed by the results obtained in Figure 6.6 where the *Power* strategy leads to the lowest average upload bandwidth utilization per peer.

[RLC08] also proposes an algorithm to dynamically improve the overlay network based on the same criteria *Power*. An improvement of this work has been proposed in [RLC09] by considering loops in the overlay. Because it is using an overlay adaptation mechanism, it is targeting mainly a network of super peers for which a failure is unlikely to happen.

In [HRV09], the authors prove the *NP-Completeness* of the problem of minimizing streaming delays in a P2P session. Then, they propose a centralized approximation algorithm that leads to a streaming delay which is at most $O(\sqrt{\log n})$ times the optimal solution. Peers are partitioned into clusters based on their regional aggregated streaming capacities. Each cluster has one head peer and all cluster heads form a virtual backbone overlay. Also, each cluster head has a virtual upload capacity equal to the aggregated upload capacity of the cluster peers. Mesh links are, then, extended within clusters. A distributed algorithm is also proposed and evaluated through simulations.

A collaborative tree-mesh overlay named *mTreebone* has been proposed in [WXL10]. Through studies of traces of an existing mesh-based P2P system, PPLive, the authors state that most of the content blocks being delivered to peers follow a tree topology or a small set of trees which are made of so called stable nodes. Therefore, *mTreebone* operates as follows. Firstly, it identifies stable nodes based on their age (their life-time duration in the streaming session). Secondly, with those stable peers, it forms a backbone tree through which most of the data will be pushed to peers. Thirdly, it constructs an auxiliary mesh overlay to link other peers to each other and to the back-bone tree. The pull-based mesh overlay is mainly used to fetch missing content and

to face peer churn. The backbone tree can be optimized to improve delays through some evolution algorithms.

However, this system is likely to suffer from a non optimized upload bandwidth utilization. Indeed, leaf nodes in the tree will only serve other peer linked through the mesh overlay which is targeting missing data and peer churn situations only. In addition, nodes in that tree may have some performance issue. For instance a stable node may have a high end-to end delay with the source or may have a low upload bandwidth. Moreover, as a peer is the only one that decides, based on a threshold value, whether it can promote itself as a stable node, there may be a situation where most of the peers join the backbone tree. Such a scenario will result in a poor mesh overlay with respect to the number of the available links which affects its resiliency.

Most of the mentioned works rely on the overlay and scheduling combinations (mesh, pull) or (tree/multi-trees, push). Tree/multi-tree based systems are known to have resiliency issues to preserve the tree structure while pull-based systems have the problem of high delivery delays. Indeed, playback delay of pull-based systems is heavily impacted by the number of hops from the source as the delay needed to perform the content fetching between two nodes (send buffer maps, send requests and send data) usually outweighs the RTT of the link. With push-pull mechanisms, the RTT of the link retrieves all its importance as the use of the pull mechanism is limited.

Compared to the mentioned systems, our work is different as:

1. We use a mesh topology but we replace neighbors by two node lists: Parents and Children. Moreover, the number of children of a node depends on its upload capacity;

2. We use an existing push-pull algorithm for content fetching.

3. We exploit more node information such as the distance to the source and the

upload bandwidth utilization;

4. We validate the simulation results using a MILP model.

4.2.2 Toward Interactivity

Delay minimization and especially delay bounding and how they are affected by source and peer characteristics have not been deeply addressed in the early literature. Indeed, according to our reading, existing work has focused mainly on proving the feasibility of streaming using P2P systems relatively to the issues of churn, bandwidth heterogeneity, free-riders, etc. In addition, works addressing playback latency were mainly focusing on scheduling mechanisms ignoring optimal overlay construction (peering strategies) due to its inherent difficulty and hardness. In general, overlays were constructed randomly or using simple strategies such as selecting the nearest peers.

Recently, some works tried to reconsider overlay construction strategies to offer systems suitable for interactive/real time streaming. In such cases, the streaming delay is very critical compared to live streaming systems where some delay can be tolerated. For instance, the delay requirement to stream a movie is relaxed compared to streaming a live event such as a soccer world cup final or compared to a video conference.

For instance, in [LLR09] the authors propose a multi-tree construction strategy based on the insights obtained from two optimization steps. The first one computes the levels of peers in the trees to minimize the delay by placing the highest uploading nodes near the source. The second one computes the best links to use between each node in level k in the tree and nodes in level $k-1$.

Unfortunately such model ignores the propagation delay by considering the hop distance only. High uploading nodes are positioned near the source. When dealing

with peers at worldwide locations, a node may have a high uploading bandwidth as well as high end-to-end delays with the source or to other peers in the next level. Clearly, there is a trade off that the hop delay estimation ignores. In our work, we take into account such a trade off by computing end-to-end delays based on the propagation delays.

[ZSXY08] proposes to guarantee delay for P2P live streaming over Internet through the utilization of a push-pull based mechanism within a static environment. The key idea is to double or to reduce the number of times a packet is sent by the source to peers without requests in order to meet the delay requirements.

In other words, we would say that the server is covering bad peering decisions by increasing its upload transfer volume to send more packet copies directly to other peers.

4.2.3 Studies with Main Focus on Content Retrieval

P2P streaming systems have inherent limitations due to constraints imposed on their topologies (tree or multi-trees) or their content retrieval mechanism (pull). Tree based systems, specially multi-trees, are known to lead to good performance. However, they suffer from resiliency issues to maintain the tree structure in face of a churn rate. Mesh based systems are more resilient to churn but rely on pull scheduling. The pull mechanism operates through three steps at each node: Sending buffer maps to neighbors, requesting content and finally sending content. The resulting delay is important and thus increases significantly the playback delay experienced by peers.

PRM (Probabilistic Resilient Multicast) [BLBS03] is a content retrieval scheme for overlay trees that focuses on improving resiliency against large failures and thus on increasing delivery ratios with low overhead and bounded delay. There are two base mechanisms. The first one is a proactive randomized forwarding where each peer

randomly chooses some peers (that are already receiving content from their parents in the tree) and forwards to them some packets with very low probability p (around 0.01) to alleviate content redundancy. This increases the delivery ratio with high probability and facilitates the repair of a subtree disconnection. Indeed, in such a case, only the root of a subtree has lost its connection to its main parent in the tree (that is not using probabilistic forwarding). Many nodes in the subtree, including the root, probably remain connected thanks to links to other parents in the initial tree. The second mechanism uses NAKs to request packets that are not received. This is made possible by including a bitmap mask of received packets with each sent packet.

Bullet [KRAV03] is a system for high bandwidth data dissemination. Applications include large-file or real-time transfer. The main objective of Bullet is to maximize the bandwidth seen by the receivers. Bullet starts with the construction of an overlay tree. On its uplinks, a node tries to send disjoint sets of data. The receiving nodes retrieve missing items by linking to new parents. The overlay becomes a mesh. The bandwidth observed by a peer is independent from the one in the underlying overlay tree. The retrieval of missing content is achieved by having each node periodically disseminating a subset of its state. The subset selection is conducted in two phases:

- Collection: Starting at the leaves and propagating towards the tree root, each node sends to its parent (in the overlay tree) a *collect* message containing a random uniform subset of its descendants. This subset is obtained through *Compacting* which takes as input the subsets sent by the children nodes;

- Distribution: starting at the root and propagating towards the leaves, each node sends to each child a *Distribute set* that contains a random subset of nodes that are disjoint with the set of the descendants of that child.

Data is transferred using an unreliable version of TCP-friendly Rate Control (TFRC) [HFPW03]. Actually, it was observed that lost packets were more easily recovered

from other peers than from retransmission mechanisms [KRAV03]. Simulations have shown that Bullet can achieve an average bandwidth that is twice the one observed in a typical overlay tree.

Coolstreaming [ZLLY05] adopts a data-centric receiver-driven approach based on a pull approach. Each node exchanges buffer maps with all its neighbors. Then, it is able to request disjoint wanted data from neighbors that possess it. Thus, a node may have different parents for each request. The main problem of this solution is that it leads to a high delivery delay because of the use of the three-step content retrieval mechanism: content advertising, request and sending. This problem was tackled in [ZLZY05], see Section 4.3, where the authors propose a hybrid push-pull approach.

In [YP05], the authors tackle the problem of frame forwarding in a tree overlay. They consider that frame-buffer mismatch between a node and its parent may lead to playback interruption due to frame loss. [YP05] also proposes a frame synchronization mechanism that tries to minimize the lag between the parent and the child nodes at the joining process. This mechanism finds the starting frame that must be forwarded by the parent with the objective that this frame will be played by the parent and the child at nearly the same time. If the frame is not available, the joining peer may wait for it. In the case of a reconnection to the tree, a peer may suffer buffer underflow or local frames loss due to significant mismatch between its own buffer and the parent's buffer. However, favoring one situation on the other does not reduce the frame loss rate. The main contribution of this work is that it is one of the rare studies that treats the problem of content discrepancy among peers in a streaming session. The main limitation lies in the fact that it is concentrated on tree topology (one parent) and thus its synchronization mechanism is not suitable for a multi-sender approach.

To cope with content discrepancy, the authors in [BCMR04] study several reconciliation techniques such as existing Bloom filter [FCAB00] approaches and recoding using erasure coding [LMSS01] as well as a proposed technique based on a tree structure combined with Bloom filters. The last technique was used in [KRAV03]. Unfortunately, all these techniques do not try to avoid the appearance of content discrepancy as they are reactive solutions. Moreover, they increase delay and complexity because of both computation and the three steps of content fetching: advertise, request and send.

rStream [WL05b] tries to eliminate content reconciliation in a multi-sender approach by using rateless encoding of the content. With high probability, only $(1+\epsilon)k$ different blocks are necessary to reconstruct original content, with ϵ being a small value near zero. To completely eliminate the need of content reconciliation, each node decodes received blocks to retrieve original content blocks that will be recoded based on the same parameters used at the source. A node only forwards recoded blocks which are almost unique. Simulations have shown that rStream achieves a high throughput rate but at a high computational cost as mentioned in [HRV09].

Using rateless coding may lead to peer waiting for the necessary number of blocks to be able to decode already received ones which affects the playback delays. In addition, precise knowledge of receivers updated buffer maps may be necessary to have a good performance.

We also think that this solution may increase, in a variable manner, the delay at each overlay node depending on resource availability and hardware/software capacities (such as the CPU and the RAM). In addition, it will consume more resources at each overlay node.

In [MR09], the authors try to propose an optimal streaming pattern over mesh overlays. They identify two performance bottlenecks: (i) A bandwidth bottleneck

that occurs when the aggregate available bandwidths of the parents is lower or higher than the incoming link access of the receiver, (ii) A content bottleneck that occurs when the data amount of useful content at the parents is below their available upload bandwidth. The proposed pattern contains two phases: Firstly, content streaming starts by a phase of segment diffusion where each peer connection to parents is exclusively reserved for periodic pushing of a new segment unit. This phase ends when peers at the higher level (farthest from the source) receive the first units of the concerned segment. Secondly, a swarming phase starts and peers pull missing data.

The author of [Liu07] derives theoretical minimum delays for P2P streaming systems, $O(log_2 N)$, where N is the number of peers in a session. He proposes a scheduling strategy called *Snow-ball* that theoretically achieves minimum average delivery delays. However, it is difficult to implement this strategy in a real environment because it assumes:

- That the server and each peer are relaying chunks at their maximum upload capacity at every time instant.
- The server and each peer know exactly the buffer map of destination peers.

In [LYHT08], the authors identify redundant traffic as the main issue with push-pull scheduling strategies and then try to address it. To that purpose, a peer divides its buffer into three sections: A pull window for packets that will be pulled, a push window for packets that will be pushed and a tolerance window (of size equal to the average RTT with its neighbors) for some packets between the two former windows to avoid redundancy. To schedule substreams, a peer assigns a sending token (number of substreams that the neighbor may send) to each neighbor based on the estimation of its upload bandwidth. These estimations are obtained from the initial buffering phase (pull scheduling). A push request will contain the substream ID and the starting

chunk ID. To address dynamic changes specially in upload bandwidths of peers, re-scheduling is done by subscribing for the concerned substream with another neighbor if the old one fails to deliver more than half of the substream. To avoid loops, rescheduling to the new neighbor is done only if it has a higher substream head than the requesting node. Simulations show that it achieves better throughput than a typical push-pull algorithm.

Several gossip-based broadcast strategies have been proposed as they improve reliability to churn and data loss. In a typical scenario, each peer sends content to a randomly selected subset of neighbors, and so forth. Different heuristics are proposed to let a peer select the content to be sent.

In [WL07], the authors propose a scheduling strategy called R^2. It is based on a random push combined with random network coding. The video stream is divided into large segments (4 seconds each), and each segment is made of blocks. Each peer is a seed. To limit the coding complexity, blocks encoded by a peer, at a same time period, belong to the same segment. For a receiving peer, blocks are equally useful independently of their seed as long as they belong to the same segment. Thus, a seed randomly picks a receiving peer and then randomly selects a segment that is needed by that peer. As there are no explicit segment requests, buffer map exchanges are needed in order to select a needed segment and to ensure there is no segment redundancy. They are done whenever there is a change in the buffer. Thanks to the big size of the segments, their exchange is not frequent. The R^2 strategy also imposes that there is a synchronized playback among all peers: They play the same segment at the same time.

However, R^2 does not investigate the impact of the discrepancy among RTT of parents of the same node. Indeed, if buffer map information do not arrive at the same time, it will lead to block redundancy. Moreover, the combination of random

push and random rateless coding was made possible due to the fact that the authors only consider streams with a low size, $64kbps$. This was necessary in order to obtain a buffer map with a low number of segments.

The authors of [BMM+08] study the optimality of several push scheduling algorithms with respect to the criteria used to pick the destination peer and the chunk to send to it. Some of the schemes achieve theoretical optima with respect to delay or to rate performance. However, those results were obtained with an overlay path length estimate using its number of hops rather than the network delays. Also, it was assumed that the overlay is a complete graph and that each sending peer has precise information about the buffer content of its neighbors. The authors did not address how to recover lost data.

For instance, a strategy called *DP/RU*, where a peer sends a randomly selected useful packet (RU) to the most deprived neighbor (DP) is analytically shown to be rate-optimal. However, the work presented in [LGL08] shows through Planetlab-based experimentations that *DP/RU* fails to achieve high rates. Indeed, it led to a large number of duplicate packets at receiving peers (collision) and to high rate of chunks missing their playback deadline. This was attributed to the fact that *DP/RU* needs correct information about the buffer maps of neighbors and that such information could be easily outdated. Also, a deprived peer may be chosen simultaneously by its neighbors so collisions will occur.

In [GLL08], an adaptive queue-based chunk scheduling (*AQCS*) is presented. Each peer maintains a forwarding queue in which it stores chunks that were pulled from the server. The server maintains a pull queue in which it stores the pull requests of participating peers. A chunk will not be sent more than once following a given peer pull request. When the pull queue is empty, the server pushes a replicated chunk to all peers in the session. These identical pushed chunks will not be relayed. Although

AQCS has been shown to achieve optimal streaming rate in a realistic environment, it relies on strong assumptions such as a fully connected mesh overlay which raises a scalability issue.

The authors of [LGL08] compare different types of scheduling strategies (*AQCS*, *DP/RU*, Random Pull, etc.) to separate situations where a simple random scheduling is sufficient to achieve good performance from the situations where more elaborated designs are required. The main results indicate that *intelligent scheduling does make a difference when the resource index is low*. However, this conclusion only applies to the performance related to the rate of missing chunks. It was showed that delays can be reduced with intelligent scheduling [LGL08].

In [PM08] the authors improve the *DP/RU* algorithm that was proposed in [BMM+08]. They replaced the packet selection mechanism: Instead of randomly selecting a useful packet, the new strategy selects the latest useful packet for the considered deprived peer. However, the authors also replaced the push mechanism in *DP/RU* by token-based pull where a peer sends a token to a potential destination peer to inform it of what chunks it can provide. The receiving peer may modify the token with a new number of chunks and/or different sequence numbers and send it back. Compared to *DP/RU*, the new *DP/LU* algorithm improves the delay performance.

In [BLPL+08], the authors propose a push scheduling where a peer relays every received packet to the maximum number of neighbors. Received packets are stored in a FIFO queue (most likely in different order than the stream one) and then pushed to neighbors. At each step, there is only one chunk to be uploaded which is the queue head. The number of neighbors that will receive the chunk is gradually updated at each step depending on whether the sending queue is empty and on the transmission time of the last step. To select the neighbors that will receive the chunk, the sending node advertises the availability of the next chunk so that neighbors send missing

notifications for the current chunk. The peer classifies interested nodes into classes based on the ratio of its own uplink bandwidth estimation to the download bandwidth estimation of each neighbor.

The authors of [ZCLB09] also propose a push scheduling in mesh overlays. Parents of a node actively forward packets to receiving child according to a pattern already set by him. The pattern contains the bitmap of a packet cycle and a starting packet ID. Lost packets are explicitly requested through backup parents that piggybacks their buffer maps with sent packets.

Although the last few works try to propose push scheduling strategies, they still rely on advertisements to let potential receiving peers know from where to get a specific piece of data. In Chapter 9, we propose a push strategy which is different from the aforementioned approach as:

1. We are not proposing a random push strategy as we do not randomly pick a packet to be sent or a neighbor to send to;

2. We eliminate the need of buffer map exchanges. Specific requests are only made for missing packets and are sent to all or to a subset of the parents. At no time, we need to know the buffer map of a node;

3. We eliminate chunk/packet scheduling. If there is an available space in the upload buffer, a packet is pushed as soon as it is received by the concerned parent. We do not impose the substream that will be scheduled by a peer: It will subscribe to the first received one;

4. We impose the fact that each node controls the acceptance of incoming requests (push or connection).

4.3 GridMedia

Description. GridMedia [ZLZY05, ZZT+05, ZZSY07] is a large-scale live media streaming system. We present this system in more details because its push-pull mechanism is being used in several parts in our work as it will be seen in the coming chapters.

Gridmedia uses an unstructured mesh overlay based on a gossip-like protocol and a streaming management based on a combined push-pull approach. A joining node contacts a rendez-vous point and gets a number of random existing nodes. Then, it selects some nodes with minimum RTT and some other nodes randomly to enhance the connectivity of the overlay.

The stream is divided into n substreams according to the sequence number of packets. Contiguous packets from different substreams form a packet group. A set of k contiguous groups forms a packet party. Each group has a group ID from 0 to $k-1$.

For instance, the group with ID equal to 0 is chosen as the base to take push scheduling decisions, i.e., requesting a push subscription from one peer or canceling it. The pusher of a substream is the parent from which the node has successfully pulled a packet that belongs to both the wanted substream and a group with ID 0. This way, the switching between pushers is not done too frequently.

At startup, a node requests packets using the pull mechanism with random scheduling until receiving packets from specific substreams. Then, it subscribes to the sender so that the latter relays the packets of that substream directly to it. If the receiving quality of a node is greater than 95%, it does not request buffer maps anymore.

Lost packets are recovered the same way. If a streaming packet has not been pushed and has been pulled by another node, then the subscription with the first node is replaced by a subscription with the new one, (always at the beginning of a

group with ID 0).

This dynamic behavior makes the algorithm able to adapt with more or less success to the overlay structure. If the overlay contains complex dependencies among nodes, cycles for instance, there is no guarantee that all packets being received are pushed. Some of them will be pulled.

The content distribution does not necessarily follow exactly the constructed overlay. A node may have 5 parents but, for instance, only 2 are active and sending the full content.

Performance Evaluation. The performance criteria used to evaluate GridMedia are as follows.

- Absolute delay: Delay between playback time at a user and sampling time at the server;
- Absolute Playback Deadline: Playback time (after which the received packet is useless);
- Delivery ratio: The number of packets that arrive at the receiver before the absolute playback time over the total number of packets;
- α-playback time: The minimum absolute delay at which the delivery ratio is $\alpha \leq 1$.

Group size	310
Max. Neighbors	5
Period of BM and RP exchange	1 sec.
Average packet rate	30 packets/s
Bit rate	310 Kbps

Table 4.1: GridMedia Experimentation Setup

The playback delay depends on the ratio r of the received packets to the needed packets within buffer intervals of one second. Every two consecutive intervals have a time slot lag of $250ms$. The playback delay is then determined by the lowest i^{th} time slot for which the corresponding buffer interval satisfies $r \geq 99\%$. The playback delay is then equal to $i \times 250ms$, i starting at 160 downward 0. Zhang *et al.* [ZLZY05], present some experimentation results (based on the setup of Table 4.1) with no uploading limit for each node.

In static environment, (zero churn rate), with the proposed pure pull mechanism, the average 0.97-playback time is 10 seconds. In the proposed push-pull mechanism, it drops down to 2-3 seconds. The last node which is supposed to be far away from the source took 20 seconds to see its average 0.97-playback dropping to the average.

In dynamic environment, with on-line and offline duration per node exponentially distributed and respectively of an average of 100 and 10 seconds, average 0.95-playback is 22 seconds for pull approach and 13 seconds for the combined push-pull mechanism. Control overhead which is defined as the ratio of control traffic to total traffic for each node is less than 2%. It includes keep and probe messages between neighbors, member table packets, buffer map packets, request packets and push management packets. Overhead in push-pull approach is less than pure-pull approach because there are less requests between nodes.

Link stress performance has also been measured. It refers to ratio of total packets passing through that link over the average packet rate at the streaming server.

Experimentations show that 50% of the links have a link stress less than 0.5 and that only 1% have a link stress greater than 4.

In the case of a limited upload ($500kbps$ in the experimentations) the delays become larger. Thus, in static environment, pull and push-pull achieve respectively 18 and 13 seconds. In dynamic environment, it rises to 24 and 20 seconds. Link stress is

globally the same as in the unlimited case.

4.4 Technical Issues

Despite the amount of works that have been conducted to address P2P streaming issues, P2P streaming systems still face several challenges:

Correlated objectives. There are many objectives that were tackled by existing work on P2P live streaming in the phase of overlay construction. However, it is not easy to define an optimal solution as several tradeoffs must be made to reconcile many criteria. We observe that a non negligible set of studies that tackle resiliency and throughput maximization were implicitly targeting the design of a P2P streaming that can face issues such as peer churn and bandwidth heterogeneity and asymmetry. On the contrary, an approach that focuses on delay minimization can be seen as having an objective to provide a given QoS to end-clients. Moreover, delay is correlated with other objectives. Indeed, weak resiliency and low throughput will imperatively result in delay increase due to time and resources that are spent in repairing failures (reconnecting peers or selecting new parents) and in getting the content. Maximizing resiliency and throughput must not be seen as an objective but as a mean to provide a QoS such as delay minimization.

Lack of Delay-centred approach. Most of the existing works do not provide a complete approach to minimize source-to-peer latency. Generally, they try to bound the delay to be acceptable and to take some related intuitive decisions. In addition, they do not minimize delay based on both the overlay construction/maintenance and the content retrieval mechanism, although we think that better results could be achieved. Also, there is no accurate delay estimation: Existing studies usually model

the delay as the minimum or maximum number of hops between the source and the node depending on the optimized criteria. Hence, we state that several research issues related to better delay management are still to be addressed.

Lack of complementarity between content distribution and overlay construction. When targeting an objective, most of existing works focus on the content distribution mechanism or the overlay construction protocol. Actually, these two parts are complementary and considering both of them will help to achieve better results.

Peers content discrepancy. This issue occurs due to networks and overlay characteristics, like data loss, bandwidth heterogeneity, churn rate and overlay maintenance (reconfiguration, reconnection...). Nodes will have significant different working sets, i.e., buffer content. If parents of a same node experience such discrepancy, it will result in higher delay or playback interruption at the receiver side. Although content reconciliation techniques have been proposed [BCMR04], they are time and resource intensive.

Reactive fetching strategy. Content reconciliation is commonly used in multi-sender based approach. We consider that it is a reactive approach to the issue of working set discrepancy. The problem is that it results in more delay and overhead. The use of a rateless encoding, see [WL05b], can be seen as a proactive approach but delay is increased because each node must receive enough blocks before decoding them and then recodes original blocks before sending them to another peer. A proactive approach that deals with a content distribution pattern and a local fetching strategy (between a peer and its parents) will help eliminating the need of content reconciliation and thus, will save delay and resources.

Overlay and Application Performance. Existing overlay and application performance metrics present a high diversity. In addition, sometimes a metric with the same name may have different definitions such as "Delivery ratio" in [ZLZY05] and [BB04]. Consequently, it is difficult to really evaluate a system or conduct a comparative study. In addition, as it is the case in any streaming system, there is no quality evaluation by an end-user. However, metrics such as delay and playback continuity can be good indicators.

Content advertising. In most content retrieval mechanisms used with mesh overlays, a content advertising is needed so that a receiving peer knows from where to get specific blocks of data. Therefore, the performance of such strategies relies on the accuracy of the exchanged buffer maps to send useful data while avoiding redundancy. In addition, the delay and the overhead being incurred are still significant.

4.5 Deployment Issues

In this section, we present some issues related to existing works with respect to the expected enhancements of the streaming needs in the future.

Resource Exploitation. Usually, existing P2P streaming systems set the source transmission rate to hundreds of *kbps*. This is the case in both Internet oriented [PPL10, PPS06, Sop10] or academic-oriented [ZLZY05] P2P streaming applications. Moreover, based on existing codecs, this range of rate is unable, to compete with the quality of the traditional terrestrial TV broadcast for example. However, Internet access keeps evolving and thus providing both increased upload and download capacities. Therefore, multimedia services are getting more and more resources. This leads us to say that we have to formulate the streaming problem based on the percentage

of bandwidth utilization at the end-user instead of choosing specific rates.

Traffic of Internet Service Providers ISP(s). ISPs are worrying about their networks being flooded with the costly P2P traffic specially cross-ISP traffic. Subsequently, many ISPs are throttling known P2P protocols such as BitTorrent [BCC+06]. Thus, it is necessary for a P2P streaming protocol to make efficient use of the bandwidth resources. New proposals call for ISP-friendly P2P protocols.

Interoperability Between Protocols. The crucial point that makes the P2P streaming architecture an attractive solution is the number of participating peers. The problem is that there are many protocols and related based client applications that are available both in Internet and academic environments. Hence, if a same live event is broadcast by many streaming sources using different protocols, the users will be splatted according to the application protocol they are using. This may affect the streaming quality if there are not enough users within a particular session.

Part II

Overlay Construction

Chapter 5

Trade-offs in Peer Delay Minimization for Video Streaming in P2P Systems

5.1 Introduction

Peer-to-Peer (P2P) architecture is considered as an attractive and scalable solution for content distribution which does not require Internet infrastructure changes and which helps eliminating bandwidth bottleneck at the content source server. Applications include content delivery such as file sharing (see, e.g., Gnutella [Rip01], etc.) and streaming (see, e.g., pplive [PPL10], sopcast [Sop10], etc.). Nevertheless, specially for live video streaming, P2P systems face many challenging issues such as efficient and optimized overlay construction [PWCS02, CDK+03, KRAV03], content retrieval mechanisms [ZLLY05, ZLZY05, MR06], and suitable video coding schemes [PWCS02, WL05b]. Most of these works focus mainly on maximizing throughput or maximizing resiliency while trying to bound source-to-end delay which are estimated by the number of hops from the source.

Little work has been done to investigate the impact of the source and peer characteristics on playback delays. Such characteristics include the video streaming rate,

the upload bandwidth of the nodes including the source, the maximum transmission rate over an overlay link, the end-to-end network delays, etc.

In the present study, we are interested in providing a live video streaming session to the maximum number of end-users using a unique streaming source at a constant bit rate. We believe that low playback delays are a key issue in live video streaming. Thus, without a study of how they are affected by the aforementioned characteristics, one cannot have a clear view on how a P2P system will perform.

This chapter describes a linear optimization model for delay minimization at every peer in the streaming session. Through the study of a static model (no churn rate), we would like to investigate the key challenges in order to achieve the best performance before focusing on the dynamic aspects of a real P2P network.

The chapter is organized as follows. In Section 5.2, we detail the motivations of the present work. In Section 5.3, we describe the linear programming model that is based on upstream peer delay balancing at each peer of the P2P system. Numerical results are presented in Section 5.4 and conclusions are drawn in the last section.

5.2 Motivation

The performance of a P2P streaming system is tightly related to the characteristics of the video being streamed (mainly the streaming rate) and to the upload capacity of participating peers. The performance can be relative to the perceived video quality and the playback delay. In this work, we are concerned by playback delay performance and how it is affected by peers and source characteristics.

Unfortunately, the fact that there is no global or preliminary knowledge of participating peers may lead to some problematic situations. For instance, the content

provider of a P2P system must choose in advance the streaming rate of the video. Depending on the upload capacity of upcoming participating peers, the selected streaming rate may be too high or too low. In the former case, the P2P system will have difficulties to handle the intended streaming rate. Thus, playback interruptions will be frequent and new connecting peers may face a denial of service. In addition, playback delays will be higher. In the case of a low streaming rate, the potential of the P2P system may be under-exploited as it is not taking full advantage of its upload capacity. Clearly, there is a trade off to make.

Before designing and deploying a P2P system, we need to study how playback delay can be affected by the source and participating peers resources in order to avoid streaming quality issues and to better exploit the P2P system.

5.3 OCDB problem

In this chapter we make use of a mixed integer linear programming model to conduct our study. The objective of the Overlay Construction with Delay Balancing (OCDB) problem is to establish the best compromise between the minimum delay and a new criteria that we call upstream peer delay balancing. As a content retrieval mechanism is complementary to the overlay construction, delay balancing ensures that a node chooses, as much as possible, parents with similar playback delays.

5.3.1 Assumptions

In the optimization model we propose, we assume that the playback rate at end-nodes is always equal to the streaming rate, i.e., there is no playback interruption. This is a realistic assumption as even with data loss there are some correction techniques (redundancy or interpolation) that compensates for data loss. Finally, at this stage,

we do not take into account the churn rate.

5.3.2 Notations and Definitions

Let V be the set of peers in the network. Let P_v and C_v be the parents and the children of a peer v, respectively. S is the streaming source with rate R. We have $S \in V$ and P_S is empty. Denote by $V^\star$ the set of nodes deprived from the source node: $V^\star = V \setminus \{S\}$.

Denote by b_v^{UL} (respectively b_v^{DL}) the available upload (respectively download) bandwidth of node v. We have:

$$0 \leq b_v^{\mathrm{UL}} \leq \overline{b}_v^{\mathrm{UL}} \quad \text{and} \quad 0 \leq b_v^{\mathrm{DL}} \leq \overline{b}_v^{\mathrm{DL}} \qquad v \in V.$$

Upper bounds, $\overline{b}_v^{\mathrm{UL}}$ and $\overline{b}_v^{\mathrm{DL}}$, come from the performance characteristics of the peers.

Let d_{uv}^N be the end-to-end latency, at the physical network level at node v relatively to node u for $u, v \in V, u \neq v$.

5.3.3 Variables

We have 5 different vectors, one state vector (e), one rate transmission vector (r), and three delay vectors ($d^A, d^{P,\max}, d^{P,\min}$).

Let $e \in V \times V$ be a state vector such that $e_{uv} = 1$ if there is an overlay link from node u (parent) to node v (one child of u), $u, v \in V$, $u \neq v$ and $v \neq S$, i.e., u transmits data directly to v and $e_{uv} = 0$ otherwise.

Next, let r be the transmission rate vector where each component r_{uv} denotes the transmission rate from node u to node v, for $u \neq v$. r must satisfy the following relations:

$r_{uv} \geq 0$,

$e_{uv} = 0$ if and only if $r_{uv} = 0$,

$e_{uv} = 1$ if and only if $r_{uv} > 0$.

The third vector d^A corresponds to playback delays, i.e., d_v^A is equal to the playback delay of v relatively to S, $v \in V^\star$.

$$d_v^A \geq \max_{u \in P_v} (d_u^A + d_{uv}^N)$$

Finally, we have two more vectors to measure the largest and the smallest parent delay at every peer. Components $d_v^{P,\max}$ and $d_v^{P,\min}$ of these vectors measure the largest and the smallest playback delays experienced by the parents of v, respectively. Indeed,

$$\begin{aligned} d_v^{P,\max} &= \max_{u \in P_v} d_u^A, \\ d_v^{P,\min} &= \min_{u \in P_v} d_u^A. \end{aligned}$$

At last, we have two more variables in order to later linearize the objective function which we will define in the next section. Let $d^{A,\max}$ and σd^P be two variables which measures the maximum playback delays, and the maximum discrepancy of playback delays among the parents of a given child:

$$\begin{aligned} d^{A,\max} &= \max_{v \in V^\star} d_v^A, \\ \sigma d^P &= \max_{v \in V^\star} \left\{ d_v^{P,\max} - d_v^{P,\min} \right\}. \end{aligned}$$

5.3.4 Optimization Criterion

We propose to consider an objective which is a combination of the playback delay, d_v^A, and of the delay balancing, for each node in the overlay network. Moreover, for each criterion, we propose to consider a weighted sum of the maximum value and of the

mean value in order to overcome the drawback of each objective (no control on the different values in the case of the maximum value, no control on the extreme values in the case of the mean value). It can be written as follows:

$$f^{\text{OBJ}} \quad = \quad 0.5 \left(d^{A,\max} + \frac{\sum\limits_{v \in V^\star} d_v^A}{|V| - 1} \right) \quad + \quad \alpha \quad \times \quad 0.5 \left(\sigma d^P + \frac{\sum\limits_{v \in V^\star} (d_v^{P,\max} - d_v^{P,\min})}{|V| - 1} \right)$$

where α denotes a scaling factor between the two components of the objective.

5.3.5 Mathematical Model

The mathematical model is presented in Figure 5.1. It uses M, a large constant such that constraints where M appears become redundant if p is not a parent of v. In that case, those contraints have no impact on the optimization process. In the context of the current model, M can be set, e.g., to the duration of the streaming session.

5.3.6 Constraints

Delay Balancing Constraints. Constraints (1) express that $d^{A,\max}$ is an upper bound on the playback delay of any node $v \in V^\star$. Constraints (2) state that a source-to-node streaming delay is greater than the maximum playback streaming delay of its parents increased with the parent-child network delay. Independently of the numerical results, we can see that these constraints will yield overlays that are directed acyclic graphs. Eliminating cycles improves overlay performance by avoiding inter-dependency between nodes. Constraints (3) and (4) give an upper and a lower bound of the playback delays of the parents of a node. Constraint (5) gives an upper bound of the delay discrepancy of parents of a given node. In the remaining of the thesis, we will also refer to that delay discrepancy as the delay variance of a node.

$$\min f^{\text{OBJ}} \qquad \text{subject to:}$$

$$d_v^A \le d^{A,\max} \qquad v \in V^\star \tag{1}$$

$$d_v^A \ge d_p^A + d_{pv}^N - M(1 - e_{pv}) \qquad v, p \in V, \; p \ne v, v \ne S \tag{2}$$

$$d_v^{P,\max} \ge d_p^A - M(1 - e_{pv}) \qquad p \in V, v \in V^\star, p \ne v \tag{3}$$

$$d_v^{P,\min} \le d_p^A + M(1 - e_{pv}) \qquad p \in V, v \in V^\star, p \ne v \tag{4}$$

$$\sigma d^P \ge d_v^{P,\max} - d_v^{P,\min} \qquad v \in V^\star \tag{5}$$

$$\sum_{c \in V^\star \setminus \{v\}} r_{vc} \le \overline{b}_v^{\text{UL}} \qquad v \in V \tag{6}$$

$$\sum_{p \in V \setminus \{v\}} r_{pv} \le \overline{b}_v^{\text{DL}} \qquad v \in V^\star \tag{7}$$

$$\sum_{p \in V \setminus \{v\}} r_{pv} \ge R \qquad v \in V^\star \tag{8}$$

$$\underline{r}\, e_{uv} \le r_{uv} \le \overline{r}\, e_{uv} \qquad u, v \in V^\star, u \ne v, \tag{9}$$

$$\underline{r}\, e_{Sv} \le r_{Sv} \le R\, e_{Sv} \qquad v \in V^\star \tag{10}$$

$$e_{uv} \ge 0 \qquad u \in V, v \in V \setminus \{u\} \tag{11}$$

$$r_{uv} \ge 0 \qquad u \in V, v \in V \setminus \{u\} \tag{12}$$

$$d_v^A \ge 0 \qquad v \in V \tag{13}$$

Figure 5.1: OCDB Linear Model

Rate vs. Bandwidth Constraints Constraints (6) state that a node $v \in V$ can not upload more than its maximum capacity. Similarly, constraints (7) state that each node $v \in V^\star$ can not download more than its maximum capacity. Constraints (8) state that each node $v \in V^\star$ has enough download capacity to accommodate the video streaming rate.

Performance Constraints The goal of these constraints is to help constructing an efficient and resilient overlay network. Constraints (9) express conditions on the transmission rate from a parent p to a child node v. This rate is at least $\underline{r}$ to avoid useless parents (transmission rate equal to 0) or fragmentation of the rate assignment (parents with a small fraction of the transmission rate) and, at most $\overline{r}$ to avoid relying mainly on one parent.

Constraints (10) state that the transmission rate from the source to any other node v is at least $\underline{r}$ and at most R: here, we allow the source to send at full streaming rate to one peer because, if the source fails, the P2P session will be over.

5.4 Numerical Results

In this section, we describe the results obtained through the solutions of the OCDB model. We used CPLEX [CPL10] to solve the MILP (Mixed Integer Linear Program) associated with the OCDB model. For more clarity, we present here the results for 8 nodes including the source. We start by detailing the data set used for the experiments. Then we explore and comment the different results. The resulting overlay network with default parameters is shown in Figure 5.2. For each node in the overlay, we show its identification, the value of its parents delay discrepancy (between parenthesis) and its playback delay (on the right side of the figure). Delays and variances are expressed in milliseconds (ms). We do not show the values of

Table 5.1: Bandwidth Characteristics of Nodes

Node ID	Upload BW ($kbps$)	Download BW ($kbps$)
0 (source)	900	N/A
1	200	500
2	200	500
3	200	500
4	300	500
5	400	1000
6	500	1000
7	600	1000

transmission rates on the overlay links to preserve the readability of the figure. The upload and download bandwidth capacities (in $kbps$) for each node are provided in Table 5.1. We also provide a delay table, Table 5.2, where we assume that the end-to-end delays are the same between any pair of nodes in both directions. In order to build Table 5.2, we have made use of some typical values obtained through pinging some websites (such as yahoo, ebay, google, ...) located worldwide and observing the average RTT values as they are less prone to variations. In the next chapters, more realistic delay values will be used.

5.4.1 Data set

Unless explicitly stated, the parameters R, $\underline{r}$, $\overline{r}$ and α, take the default values $442kbps$, $50kbps$, $250kbps$ and 0.5, respectively. We have determined, through experiments, that $R = 442kbps$ was the maximum streaming rate that led to a feasible solution using the described data values. We will now evaluate the performance of the proposed model in such extreme situations.

Table 5.2: Peer-to-Peer Network Delays

Nodes	1	2	3	4	5	6	7
0	20	50	110	140	150	180	200
1	-	50	60	145	155	190	195
2	-	-	20	160	200	110	130
3	-	-	-	40	45	260	255
4	-	-	-	-	20	270	280
5	-	-	-	-	-	290	300
6	-	-	-	-	-	-	30

Table 5.3: Impact of the Transmission Rate Limit, $R = 442kbps$

$\overline{\mathbf{r}}$	**221**	**250**	**350**	**442**	**450**
Max. Delay (MD)	665	665	605	600	600
Av. Delay (AD)	414.3	380	405	400	400
Max. Variance (MV)	300	310	210	200	200
Av. Variance (AV)	159.3	130.7	136.4	110	110

5.4.2 Impact of the limitation of transmission rate between two nodes

In order to explore the impact of the limitations of the transmission rate between two nodes ($\overline{r}$ in the OCDB model), we have varied $\overline{r}$ below and over the streaming rate. In Table 5.3, we present the most significant results. We conclude that the optimal solution depends on the value of $\overline{r}$. For $\overline{r} \geq R$, a case where there are no constraints on the transmission rate over an overlay link, we obtain the best performance in terms of delay minimization and parent delay variance. This is due to the fact that there is more flexibility in choosing parents and transmission rates. However, with such $\overline{r}$ values, we loose control over load balancing. We also weaken resiliency as a receiving node can take most of the content from only one parent.

Actually we can say that the interval $[350 - \varepsilon, 350 + \varepsilon]$ is likely to contain the values for which there is the best trade off between delay and variance performance on the one hand, and resiliency on the other hand. Indeed, the gain in variance and delay is very important compared to lower values (250 or 221) but it is close to the

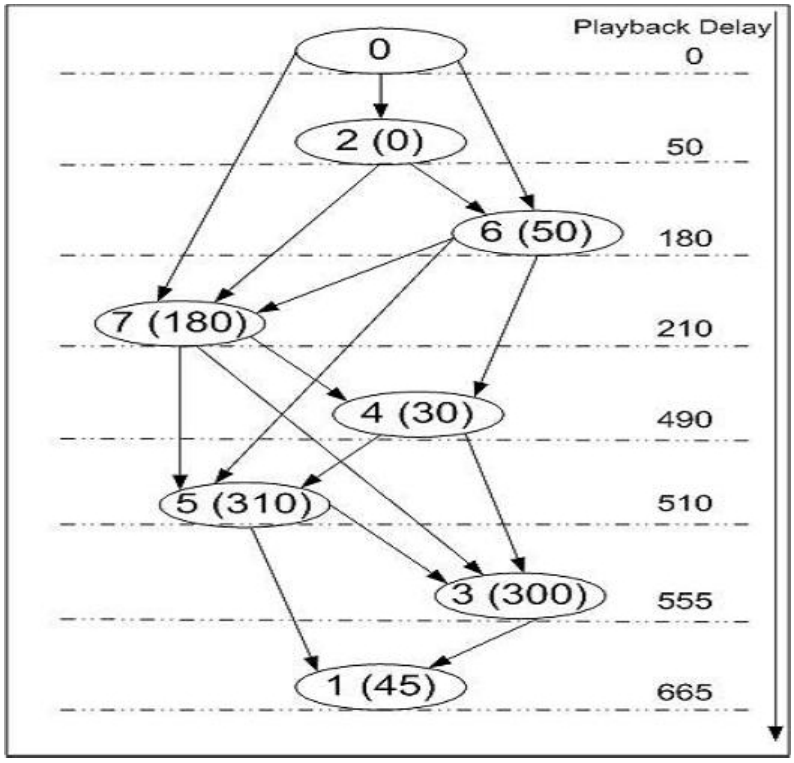


Figure 5.2: An Optimal Solution for the OCDB Problem with 8 Nodes and R=442kbps

best values (obtained with $\overline{r} = 450$). Obviously, the value 350 depends on the data set used, but for each one, such rate probably exists.

5.4.3 Impact of the Streaming Rate

In order to study the impact of the streaming rate on the performance, we used the same data set and only changed the value of $\overline{r}$ to 350 instead of the default value 250. Results are presented in Table 5.4. The main deduction is that when the streaming

Table 5.4: Impact of the Streaming Rate, $\overline{r} = 350kbps$

R	**200**	**300**	**350**	**400**	**442**
MD	180	210	290	405	605
AD	110	127.14	186.43	259.29	405
MV	0	60	55	180	210
AV	0	12.9	13.6	81.4	136.4

rate is higher, delay and variance performances degrade. This is due to the fact that when increasing the streaming rate, the amount of upload bandwidth that is used by the P2P network is higher. Indeed, the closer is the streaming rate to the maximum feasible one (after which there is no feasible solution), the worse is the performance, as we reduce the flexibility and restrict the possible choices for the parents in the overlay construction.

In addition, we note that, for some streaming rates that are not so far from the maximum allowable rate ($442kbps$) such as $350kbps$ and $400kbps$, the performance is much better:

- Reduction of 52% and 33% respectively, for the maximum delay;
- Reduction of 54% and 36% respectively, for the average delay;
- Reduction of 74% and 14% respectively, for the maximum variance;
- Reduction of 90% and 40% respectively, for the mean variance.

At the same time, the performance is not much worse than lower values of the streaming rate ($300kbps$ for example). Thus, we should avoid loading the P2P network with streaming rate that is close to the maximum feasible one. In addition, we have to look for the rate that achieves the best trade off between high streaming rate vs. low delay performance.

5.4.4 Impact of the source Upload Bandwidth

In order to discover the impact of the source upload bandwidth on the delay performance, we have modified the maximum upload bandwidth of the source, while keeping identical the overall upload bandwidth provided by the P2P network. In addition, we choose $\overline{r} = 350$ and $R = 442$. The results are summarized in Table 5.5. Here

Table 5.5: Impact of the source Upload Bandwidth, $R = 442kbps, \overline{r} = 350kbps$

$\mathbf{b}_S^{UL}$	**600**	**900**	**1200**	**1500**
MD	695	605	420	360
AD	435	405	259	212
MV	180	210	180	180
AV	88	136	97	81

again, we can see that not all the values of the upload bandwidth of the source give the best trade off for the delay performance vs. the bandwidth amount. In our case study, when the source has a capacity of 1200kbps, the P2P system reaches the best trade off as it has the best gain in delay per added source bandwidth. In addition, it is clear that increasing b_S^{UL} helps minimizing the delay because the source can serve more peers either fully or partially and thus the number of levels in the overlay will decrease.

5.4.5 Impact of heterogeneous bandwidth

To explore the impact of the non heterogeneity of upload bandwidths provided by participating peers, we assigned to each peer (not including the source) an upload bandwidth of $343kbps$ and we considered $\overline{r} = 250$. This way, the global upload bandwidth provided by the system is the same as with the initial values. Then, we did the experimentations with different streaming rates. The main results are provided in Table 5.6. The first conclusion we draw is that, as long as the streaming rate is lower than the upload bandwidth, we obtain low average delays with a low parent delay discrepancy. With a little higher streaming rate, the performance deteriorates dramatically.

Secondly, with a streaming rate $442kbps$ there is no solution which translates to a denial of service in the real case. This means that having enough upload bandwidth in the P2P system is not the sole important point. How this bandwidth is distributed

Table 5.6: Impact of Non Heterogeneous Upload Bandwidths, $\overline{r} = 250kbps$

R	**251**	**300**	**343**	**355**	**400**	**442**
MD	200	200	210	410	470	N/A
AD	124.29	130.36	127.14	153.57	201.43	N/A
MV	60	72.5	100	80	270	N/A
AV	22.86	40.71	45.71	42.14	84.29	N/A

among peers, where it is located relatively to the source and what topology is being used, are important points too. In Figure 5.2, we can see that nodes with the highest upload bandwidths are located near the source.

Based on these observations and on the results described in Section 5.4.4, we believe that most of the upload bandwidth must be concentrated near the streaming source. This will maximize the number of peers being served for a given streaming rate and will lead to better performance in terms of playback delays.

Thirdly, when comparing these results with those obtained with initial upload bandwidth values (see Table 5.7), we note that for a given streaming rate, the system with non heterogeneous bandwidth performs better in terms of delay. This is due to the fact that, in an heterogeneous situation, there is a penalty associated with putting nodes, with low end-to-end network delays (but with small upload bandwidth), or nodes, with high upload bandwidth (but with high end-to-end delays), at a specific location, specially near the source. In the homogeneous/other case, such penalty does not exist. It is as if the solution finds the best overlay in terms of delay and then assigns the rates to the links.

Table 5.7: Results with Heterogeneous Upload Bandwidths, $\overline{r} = 250kbps$

R	**343**	**355**	**400**	**442**
MD	340	420	515	600
AD	221.43	252.14	320	400
MV	180	140	180	200
AV	71.43	45	57.86	110

5.5 Conclusion

In this chapter, we have proposed a linear programming model that minimizes experienced playback delays of peers in a streaming session. Thanks to this model, we have studied the impact of the characteristics of the streaming source and the participating peers on the playback delay.

To fully exploit a P2P system and take benefits from its advantages, one needs to adjust the streaming rate and the limits of transmission rates of overlay links to suit the upload bandwidth made available by the source and the peers.

We have shown that there are many tradeoffs to be considered such as low playback delay vs. high streaming rate, low delay vs. maximum transmission rate on an overlay link or low delay vs. low source upload bandwidth.

These tradeoffs come from the fact that having a high streaming rate, a low maximum transmission rate on an overlay link or a low source upload bandwidth will make the system use more overlay links, and more specifically end client overlay links, as well as putting a higher stress on individual peers.

In this study, we have assumed a total knowledge about participating peers. In a realistic environment, we do not have such a priori knowledge which makes it more difficult to achieve an optimal performance.

Chapter 6

Toward New Peering Strategies For Push-Pull Based P2P Streaming Systems

6.1 Introduction

During the last decade, several P2P live video streaming systems have been successfully deployed in both Internet and academic worlds. The technical success of such systems comes from the fact that they are providing seamlessly enough upload resources at a low cost (since the bandwidth cost is distributed on participating peers), leveraging both cost and load on the content source server.

Available Internet systems mostly originated from China and focused on deployment simplicity while trying to satisfy acceptable video quality [PPS06, PPL10, Sop10]. Academic systems tried to push the quality further to improve other performance criteria such as playback delay and resiliency to churn rates [CDK+03, PWCS02, RS04, LN06].

Several works addressing playback delay minimization focused on content retrieval mechanisms notably through proposing push-pull mechanisms [ZZSY07, LMSW07] or improved scheduling strategies [Liu07, CXH08]. These solutions do not investigate

how to construct an overlay that helps achieving such a goal. Typically, a joining node will choose neighbors randomly or based on basic characteristics of candidate neighbors such as proximity (RTT:Round Trip Time) and upload bandwidth.

In this chapter, we show that constructing an overlay with a focus on reducing video delivery delays is a necessary step toward minimizing playback delays. Node relationships are critical for the performance. To make the right decisions, a node needs relevant information about the other peers.

With the relatively new push-pull mechanisms, the use of the pull mechanism is generally limited to the initial phase of content fetching and to the recovery of lost data. Thus, its impact on the performance of a P2P system is greatly reduced. These facts open large possibilities for comparing peering strategies in a new and suitable context and even for proposing new strategies that probably do not have a big impact on playback delays with a classical pull mechanism but recover all their influence in conjunction with a push-pull mechanism.

In the following, we propose to revisit peering strategies based on the use of a push-pull content retrieval mechanism. Our focus is to minimize the playback delay experienced by participating nodes. We propose two new peering strategies that we compare to the state-of-the-art strategies using simulation. One of these new strategies is suitable for push-pull based systems and performs better than the compared ones.

In the remaining of the thesis, the names of existing and proposed strategies will be written in *Italic* and ***Italic Bold*** respectively.

This chapter is organized as follows. In section 6.2, we describe briefly the resource management scheme we use. Section 6.3 presents the different algorithms being compared. Simulation conditions are described in Section 6.4. In Section 6.5, we discuss the simulation results. We conclude in Section 6.6.

6.2 Parent-Child based Mesh

In mesh-based P2P systems, the common practice is that each node has N neighbors. If a node is able to serve more than N nodes then its capacity is under-exploited. If its capacity does not allow him to serve N nodes fairly, then it is overloaded, which results in congestion, increased delays and packet loss.

To be able to use the upload bandwidth of a peer efficiently, we use a resource organization scheme based on the concepts of parents and children. This scheme increases the upload bandwidth utilization when needed, as each node is in full control of the utilization of its resources. We divide the upload bandwidth of a node into equal slots that will be assigned to its children.

Based on this idea, a node accepts requests to be a parent until it reaches its maximum number of used slots, then it starts refusing any request. A request may ask for one or more upload slots. Meanwhile, when a node joins the session, it keeps looking for parents until it reaches the number of needed slots or there is no new nodes to request from.

More concisely, let R be the streaming rate of a P2P live streaming session. Let P be the number of slots needed by each node in the session. In the ideal case, the upload bandwidth of each node will be divided into slots of size $\frac{R}{P}$. Denote by $U(p)$ the nominal upload capacity of a node p. Thus the maximum number of slots that p may offer is equal to:

$$\lfloor \frac{P \times U(p)}{R} \rfloor.$$

To take into account the overhead and useless utilized bandwidth (lost packets for example) we compute the maximum number of slots as:

$$\lfloor \frac{U(p)}{(1+\alpha) \times \frac{R}{P}} \rfloor.$$

where $\alpha < 1$ is a positive real number to account for the overhead and retransmissions. This way, we are likely to achieve a high success ratio of sent packets per slot.

The proposed resource organization scheme will be used in all the strategies being compared in the present thesis.

6.3 Peering Strategies

In this section, we propose different peering strategies that we compare to the strategy proposed in [RLC08].

6.3.1 Strategy Classification

According to the information being used, we classify most existing peering strategies into:

- Basic Strategies (BS) that use intrinsic and peer relationship quality information such as the upload bandwidth and the RTT;
- Partially Overlay-aware Strategies (POS) that exploit information about the status of a node in the overlay (such as its distance to the source node). They do not have global overlay information though.

BS strategies are usually simple to deploy. However, as they rely on restricted local information, they will likely fall short of achieving the best possible performance. On the opposite, POS strategies take better informed decisions that serve the global performance of the system. POS strategies result in more deployment challenges as information collection about a subset of other peers costs resources, increases delays and may lack timely accuracy.

6.3.2 Strategy Operation

As we no more deal with neighbors, a joining node focuses only on finding new parents so it may impose selection criteria that were not possible with neighbor selection.

The algorithms being presented in the sequel do not allow for eliminations of existing relationships: If a node has reached its maximum number of children, it will not eliminate one of its children to accommodate a new request. This way, we are able to make the parent selection procedure converge.

When a node is joining, it tries to get the minimum number of parents that provides it with the intended number of slots, P. This means that it gets the maximum number of slots available at each parent until reaching P slots. So a node may end up with one to P parents. Nodes with a single parent may suffer in real context, for instance, because of churn rate, node failure, performance degradation of the parent, etc. In the next chapter, we address such situation by imposing a number of parents to each node. In the sequel, the name of the strategies in the present chapter will be suffixed with MP for Minimum number of Parents.

6.3.3 The BP-MP Strategy

The *Best Parents* strategy, see Algorithm 6.1, is the first strategy we propose: A node selects the best available parents according to their distance to the source node (in ms) and to the relative RTT. The distance of a node to the source is defined as the distance of the longest path from that node to the source through all of its parents. The objective is to minimize the length of the node paths to the source. We evaluate the longest path to the source according to the method $maxDtoS$ that we propose in Algorithm 6.2. To reduce evaluating the distance indefinitely due to cycles, a node will update its distance only relatively to a parent who joined the session previously. This is a realistic heuristic as in [WXL10], the authors show that lifetime of peers in

Algorithm 6.1 Best Parent Selection

Input: L: List of potential parents of a node c
Output: LP: List of parents (may be empty) {LP has a size between 0 and P (number of needed slots)}
Begin
$c \leftarrow current\_node$
sort nodes $p \in L$, in ascending order of the maximum distance of c to the source through p: $maxDtoS(p) + \frac{RTT(p,c)}{2}$
while L is not empty and $Needed\_Slots > 0$ **do**
  $p \leftarrow Head(L)$
  if $Needed\_Slots \leq AvailableSlots(p)$ **then**
    $n \leftarrow Needed\_Slots$
  else
    $n \leftarrow AvailableSlots(p)$
  end if
  $success \leftarrow RequestSlotsFromNode(p, n)$
  if success **then**
    $Needed\_Slots \leftarrow Needed\_Slots - n$
    $LP.insert(p)$
  end if
end while
End

the session is a good indicator of their stability and thus, we suppose that a stable node has a relatively stable playback delay.

Algorithm 6.2 $maxDtoS(c)$

Input: c: Current node
Output: $maxDtoS$: Maximum distance to the source of node c in $milliseconds(ms)$
Begin
$c.maxDtoS \leftarrow 0$
for each parent node p of the current node c **do**
  if $JoinTime(c) > JoinTime(p)$ **then**
    if $p.maxDtoS + rtt(p,c)/2 > c.maxDtoS$ **then**
      $c.maxDtoS \leftarrow p.maxDtoS + rtt(p,c)/2$
    end if
  end if
end for
return $c.maxDtoS$
End

6.3.4 The Power-MP Strategy

The *Power-MP* strategy is based on a variation of the distributed algorithm proposed in [RLC08]. In the original algorithm, each peer c associates with each candidate parent p a *Power* value that is the ratio:

$$Power_{pc} = \frac{\min(RB(p), R)}{\frac{rtt(p,c)}{2} + maxDtoS(p)}$$

where $RB(p)$ is th residual upload bandwidth of node p and R is the streaming rate. Node c tries to select parents with highest *Power* until achieving the intended streaming rate. In our variation, instead of dealing with residual upload bandwidth, we deal with the residual upload slots. Thus, the *Power* of a potential parent p

Algorithm 6.3 $RequestSlotsFromNode(p, n)$

Input: p: Node to request slots from, n: Number of requested slots
Output: $SUCCESS$ if request sent, $FAIL$ otherwise
Begin
if p is not already a parent of the current node c **then**
  if p still accepts children **then**
    ask p to be a parent and request n slots from it
    return SUCCESS
  else
    skip node p
    return FAIL
  end if
else
  skip the node p
  return FAIL
end if
End

relatively to a node c is:

$$Power_{pc} = \frac{\min(AvailableSlots(p), P)}{\frac{rtt(p,c)}{2} + maxDtoS(p)}$$

6.3.5 The Rtt-MP Strategy

The *Rtt-MP* strategy is based on a variation of the ***BP-MP*** algorithm. The only difference is that the candidate nodes are sorted in the ascendant order of their RTT to the requesting node: A node will choose the nearest parents. Variations of this strategy exist in the literature.

6.3.6 The PowerRtt-MP Strategy

The *PowerRtt* strategy we propose, is similar to the *Power* strategy except that it takes into account the RTT among nodes instead of the distance to the source of a

node. Thus the *Power* of a potential parent p relatively to a node c is:

$$Power_{pc} = \frac{\min(AvailableSlots(p), P)}{rtt(p, c)}$$

Such a strategy is simple and favors trade-offs between nearest nodes and high uploading ones.

6.4 Simulation

The objective of the conducted simulations is to measure the impact of the compared peering strategies on the average playback delay observed by peers in the streaming session. The best strategy is the one that leads to the lower average playback delay.

6.4.1 Simulator

We make use of the discrete event based simulator for P2P single source live streaming built by Zhang *et al.* and detailed in [ZZSY07]. This simulator operates at the packet level and has already many useful functionalities implemented. However, it suffers from the lack of documentation. The source code of the simulator is available at [Zha09].

In the simulator, the streaming data is divided into packets of size 1250 bytes, not including headers. Some parameter values are detailed in Table 6.1. Network end-to-end latencies (RTT) between nodes are real-world values taken by default from a latency matrix computed within the Meridian project [WSS05].

For each simulated algorithm, we execute the simulation three times. The number of executions is enough to obtain relevant results because we are not considering a churn rate. The results shown are the ones computed for the last 10 seconds (this behavior is by default in the simulator). We limit each execution to a duration of

Table 6.1: Simulation Parameters

Streaming Rate	250kbps
Pull Request Window Size	125 packets = 5 s
α: Slot overhead/retransmissions	0.25
Needed Slots per Node	5

180 seconds as the system reaches a steady state well before that time. Indeed, this duration does not impact the final results as, thanks to the absence of churn, the obtained overlay is stable and each node has terminated the process of parent selection. We ensure that the results are stable by comparing them to the preceding periods (150, 160 and 170 seconds).

6.4.2 Scenario

Node Information.

We have conducted simulations for the six sets of nodes from Table 6.2. For these sets, the resource index (RI), the ratio of the upload capacity available to the needed bandwidth so that each node receives the full streaming rate, is near 1.44. We have chosen a high RI, to be able to compare the performance of the strategies free of resource constraints.

However, we believe that the RI is not precise enough as it does not take into account the overhead and the eventual retransmissions. Thus, we propose a more suitable indicator: The slot index (SI). We define the SI as the fraction of all upload slots made available by peers to the total number of needed slots for all nodes. The slots index depends on the resource index and on the α parameter proposed in Section 6.2.

It is assumed that nodes have DSL connections and that bandwidth bottlenecks are located at the edge of the network, i.e., end-node access networks [ZZSY07]. Table 6.3 shows how bandwidth capacities are distributed. The source (node 0) has

Table 6.2: Node Sets

N	51	101	201	401	701	1001
RI	1.441	1.446	1.448	1.448	1.449	1.439
SI	1.14	1.14	1.14	1.14	1.14	1.14

Table 6.3: Bandwidth Capacities of End Nodes

Download Rate	3 Mbps	1.5 Mbps	768 kbps
Upload Rate	1 Mbps	384 kbps	128 kbps
N=51	10 %	40 %	50 %
N=101	12 %	42 %	46 %
N=201	13 %	43 %	44 %
N=401	13 %	45 %	42 %
N=701	13 %	46 %	41 %
N=1001	14 %	42 %	44 %

an upload bandwidth equal to $1000kbps$. Nodes are organized into three classes based on their upload bandwidths: $128kbps$, $384kbps$ and $1000kbps$.

Join Process.

By join process we mean the way nodes join the session. The default behavior is random join. Although this is a realistic behavior, it makes it difficult to compare overlay construction algorithms since, because of the heterogeneous upload bandwidth, the order of joining of nodes has an impact on the quality of the overlay. Thus, we have implemented a deterministic join process depending on the class of nodes. The join process is described in Table 6.4. It results in an identical arrival rate for the three node types until there is no more nodes, from a specific class, to join the session.

Table 6.4: Join Process of End Nodes

Node Upload Rate (kbps)	1000	384	128
First Join Time (ms)	40	20	1
Join Period (ms)	50	50	50

Push-Pull Algorithm.

In the conducted simulations, we use the push-pull mechanism as already implemented in the simulator [ZZSY07] (see Section 4.3).

The algorithm divides the video stream into substreams based on the sequence number of streaming packets. A node requests packets using the pull mechanism until receiving packets from specific substreams. Then, the receiving node subscribes with the sender so that it relays the packets of that substream directly to it. Lost packets are recovered using the pull mechanism with random scheduling.

We introduce two modifications to the push-pull mechanism being used to make it more suitable to the simulations being conducted. Firstly, we impose that the number of substreams being pushed by a node is lower or equal to the maximum number of its upload slots. This ensures that a node will not be overloaded.

Secondly, we drop the dynamic selection of substream pushers. In the original algorithm [ZZSY07], the stream is divided into n substreams according to the sequence number of packets. Contiguous packets from different substreams form a packet group. k contiguous groups form a packet party. Each group has a group ID from 0 to $k-1$. For instance, the group with ID equal to 0 is chosen as the base to take push decisions. The pusher of a substream is the parent from which the node has successfully pulled a packet that belongs to both the wanted substream and a group with ID 0. With these two behaviors, the content distribution does not necessarily follow exactly the constructed overlay. A node may have 5 parents but, for instance, only 2 are sending the full content.

To resolve this issue, we deactivated this dynamic selection. Such deactivation would be problematic if there are cyclic dependencies in the overlay. The original push-pull algorithm avoids cycles of length equal to two links but there is no guarantee to avoid cycles of length higher than two.

With the adopted join process, we ensure that each node finds its needed slots at his first parent selection procedure so that no cycles may occur and thus the static push-pull variation will work without troubles. Moreover, this way we ensure that each selected link in the overlay will be used. Thus, the performance of the compared strategies depends totally on the constructed overlay and no more on the pusher selection made by the push-pull algorithm.

6.5 Results

In the following, we compare the presented strategies in the best possible context (no churn) as our focus is on how to construct a good quality overlay. In addition, we are not considering jitter for the delivery of packets. Studying the impact of such phenomena is left for future work.

In this section, we present and comment the results obtained through the simulations. For each figure, the number of nodes is indicated on the X-axis (51, 101, 201, 401, 701. and 1001). We made sure that the presented results are measured when all streaming packets are pushed, i.e., the system is not using the pull mechanism as in the beginning of the session. This is made possible because we assume that there is no packet loss due to network conditions. Thus, the overhead has no significant value.

6.5.1 Total Upload Utilization

The total upload utilization is the utilization ratio of the upload resource of the P2P system, (see [ZZSY07] for more details). The results of the total upload utilization of the P2P system are presented in Figure 6.1. Some curves are difficult to identify as they are overlapping, since many strategies have identical curve points. The first

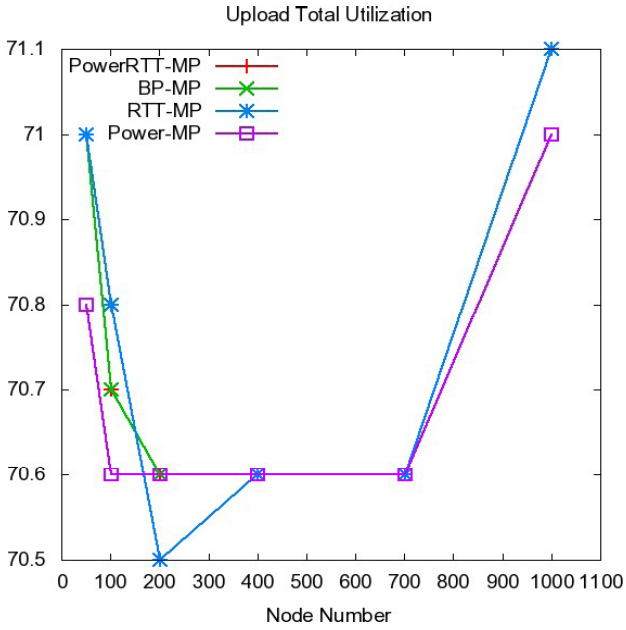


Figure 6.1: Total Upload Utilization - MP

conclusion we draw is that the total upload utilization is almost the same for all compared strategies: The maximum difference is about 0.3% or less for the same number of nodes. This is a very important result as it means that the compared strategies have almost the same efficiency with respect to resource utilization. Thus, we are able to state that the difference of performance of the compared strategies is not related to the upload resource utilization but rather to the quality of the strategy itself: If a strategy outperforms another one while using the same ressource, then the first one manages to create a better overlay network with resepect to the objective of minimizing playback delays.

6.5.2 Average Playback Delay

We define the average playback delay as the average of playback delays of all nodes except the source. We keep the method of evaluating the playback delay of a node as it is described in [ZZSY07]. The playback delay depends on the ratio r of the received packets to the needed packets within buffer intervals of one second. Every

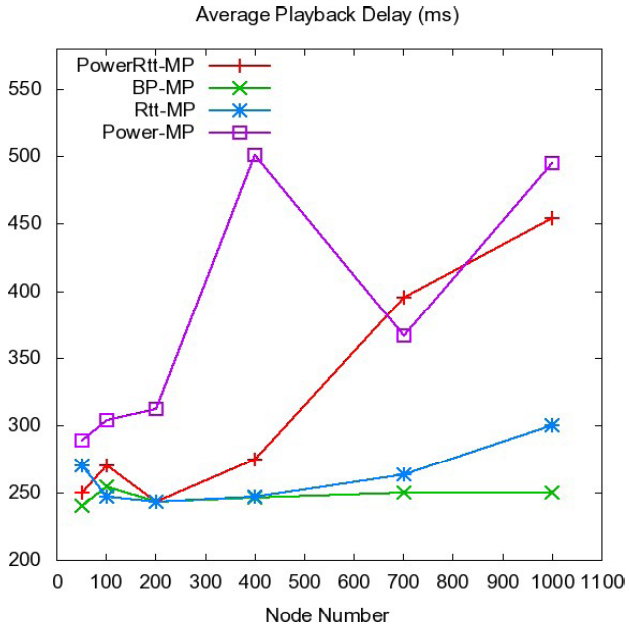


Figure 6.2: Average Playback Delay - MP

two consecutive intervals have a time slot lag of $250ms$. The playback delay is then determined by the lowest i^{th} time slot for which the corresponding buffer interval satisfies $r \geq 99\%$. The playback delay is then equal to $i \times 250ms$, i starting at 160 downward 0.

Figure 6.2 depicts the average playback delays of the compared strategies. We observe that the ***BP-MP*** and *Rtt-MP* strategies perform better than their power based counterparts *Power-MP* and ***PowerRTT-MP***. This means that power based strategies are taking some peering decisions that are not optimal. One explanation lies in the way the *Power* value is calculated as a fraction: Some side effects may occur and lead to poor decisions. For instance, a peer may select one parent with a high delay because it has a high available upload bandwidth.

Actually, by using *Power* strategies, there is a sort of gambling about the future decisions that do not prove to be right in several occasions. Thus, such strategies lack result stability: Performance may vary significantly from one scenario to the next. For instance *Power-MP* performs better with 701 nodes than with both 401 and 1001

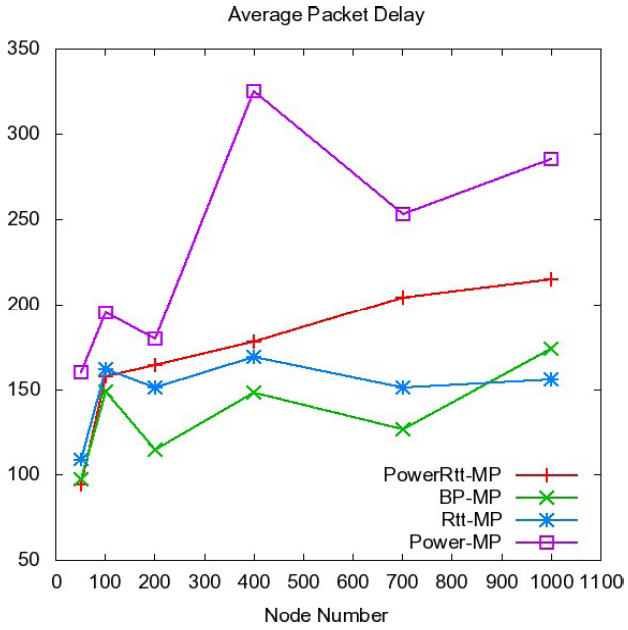


Figure 6.3: Average Packet Delay - MP

nodes.

On the opposite, ***BP-MP*** and *Rtt-MP* have a more systematic greedy behavior that consists in taking the maximum quantity of content from the best parents. The criteria in these strategies do not suffer from side effects as they are based on one parameter only.

BP-MP starts to perform better than *Rtt-MP* starting from a medium number of nodes, greater than 400. This means that, the distance to the source is a relevant criterion only when paths to the source start to have different enough lengths and high enough costs.

If it was not for our resource reservation scheme which was proposed in Section 6.2, ***BP-MP*** would probably lead to a poor performance. Indeed, this strategy assigns high packet forwarding load to the best nodes and less to the worst. So a good node may be overloaded if there is no upload resource organization scheme.

6.5.3 Global Average Packet Delay

The average packet delay of a packet is the average of delivery times needed for that packet to reach to all nodes after its creation at the source. The global average packet delay is obtained by computing the mean of average packet delay of all nodes except the source. It is calculated with the same manner as in [ZZSY07]. Each node maintains its own average packet delay for packets that have been successfully received. The average packet delay at a node is a combination of the scheduling delay (due to the push and mainly pull mechanism) and of the overlay path length. When the pull mechanism is not being used, all packets are being pushed, the average packet delay is almost equal to the average overlay path length.

From Figure 6.3, we can see that having a low global average packet delay is not enough to guarantee a low playback delay. The reason is that the packet delay deals with the successful reception of individual packets. Thus, it does not consider the contiguity of the received packets while the playback delay does. The contiguity of received packets is not guaranteed because substreams may take paths with significantly different lengths. As a consequence, a node may have parents with significantly different playback delays. For instance, for the scenarios with 701 nodes and 1001 nodes, ***PowerRTT-MP*** and *Power-MP* have similar playback delay performance although ***PowerRTT-MP*** has a better average packet delay.

To explain this result, we propose the concept of variance. The variance of a node is the difference between its highest parent streaming delay and its lowest parent streaming delay. Figure 6.4, which depicts the average variance of nodes in the session, shows that for both scenarios (701 and 1001 nodes) ***PowerRTT-MP*** have an average variance significantly worse than *Power-MP*. Thus, the advantage of having a low packet delay has been canceled by having a high variance.

Another example lies in the fact that, although ***BP-MP*** usually has the lowest

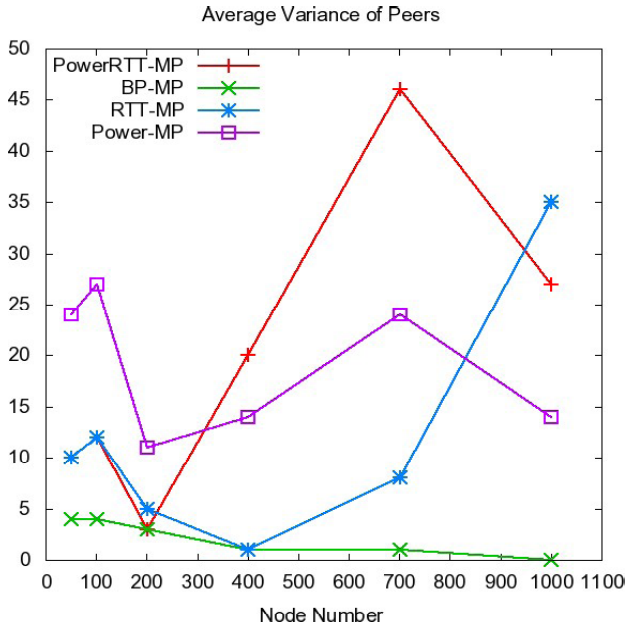


Figure 6.4: Node Average Parent Variance - MP

average packet delay, we can note that for the scenario with 1001 nodes, ***BP-MP*** has a higher average packet delay than *Rtt-MP* but still performs better with respect to the average playback delay(250 vs. 300). The reason is given by Figure 6.4 where *Rtt-MP* has a much worse variance.

6.5.4 Maximum Playback Delay

The maximum playback delays obtained for each strategy are drawn in Figure 6.5. The maximum delays obtained with ***PowerRTT-MP*** are the same as those of *Rtt-MP* except for the scenario with 51 nodes where it is the same as the ***BP-MP*** strategy. *Power-MP* performs the worst because it leads to long paths from nodes to the source. This is observable through Figure 6.3 where this strategy has significantly higher packet delays than the others. Our proposed strategy, ***BP-MP*** performs the best because it leads to the shortest paths to the source. Such a claim is confirmed by Figure 6.3 where ***BP-MP*** usually has the lowest average packet delay.

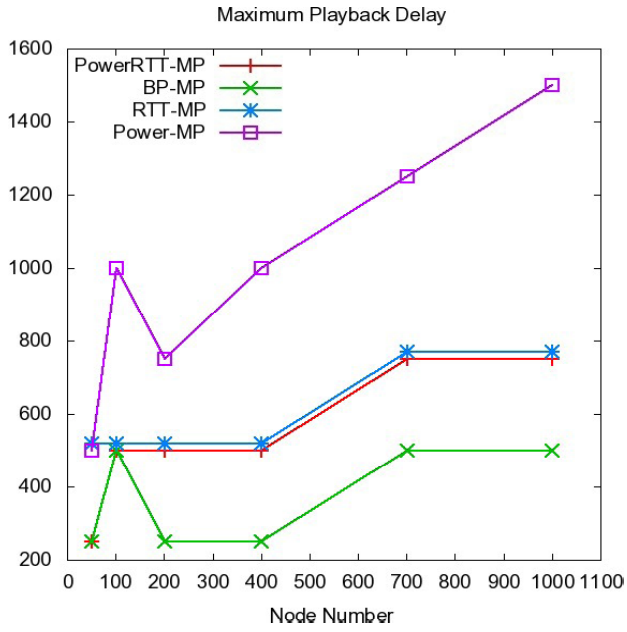


Figure 6.5: Maximum Playback Delay - MP

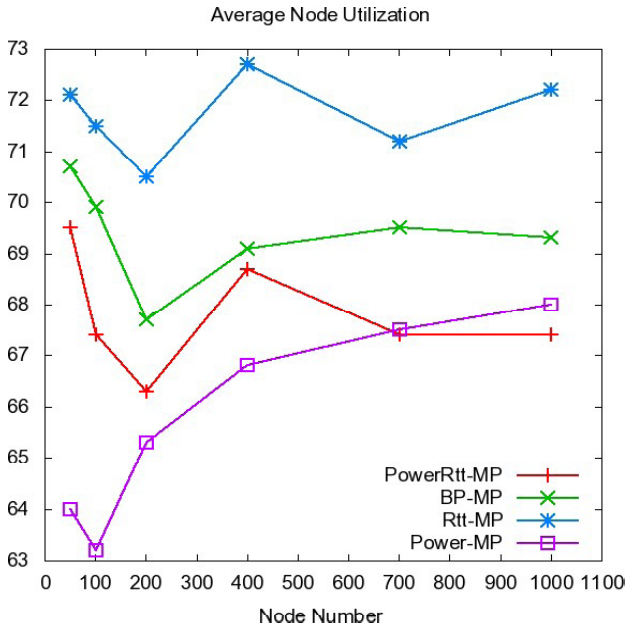


Figure 6.6: Average Node Upload Utilization - MP

6.5.5 Global Average Node Upload Utilization

The global average node upload utilization is the mean of the average node utilization as computed in [ZZSY07]. The global average node upload utilization results are given in Figure 6.6. We note that *Power* based strategies have the lowest node utilization ratio. Since these strategies are involving the available upload slots at nodes, it leads to a balanced load distribution among nodes. Another reason lies in the fact that nodes at the edge of the mesh overlay are likely to be less involved because they have have high delays.

The two other strategies are likely to impose high load on better nodes. For instance, the ***BP-MP*** strategy, having the highest node utilization according to Figure 6.6, gives preference to nodes near the source to be selected as parents. So it has a high forwarding load. The upload resource scheme we proposed in Section 6.2 ensures that the node is not overloaded.

6.6 Conclusion

In the present chapter we have compared several peering strategies for P2P video live streaming based on a push-pull scheduling strategy in a mesh overlay.

The significant difference among average playback delays of the compared strategies proves that the overlay has an important impact on the performance. Thus, constructing a good, i.e. low weight, overlay is a necessary step toward minimizing playback delays.

With respect to playback delays, one of the strategies we propose, ***BP-MP***, (Best Parents, Minimum Parents) performed the best under some scenarios and outperformed a recent strategy [RLC08] targeting reducing playback delays in mesh overlays. ***BP-MP*** is a POS strategy exploiting partial overlay information describing

the overlay distance of a node to the source.

The MP variations have the advantage to lead to simple overlays. Indeed, the number of overlay links is small thanks to a small parent degree per node. This had a positive impact on the playback delays.

In the undertaken simulations, the constructed overlays had no overlay cycles and thus all the streaming packets were pushed. Therefore, we can state that, in such a situation, each considered strategy was performing at its best, i.e., there is no other situation where it can lead to a better playback delays for the considered P2P streaming scenario.

Unfortunately, such an ideal situation is unlikely to happen within a realistic environment with churn rate and packet loss. To protect itself, each peer needs to increase its number of parents/neighbors and to distribute the load of the video stream forwarding among them.

Subsequently, in order to improve the proposed *MP* variations and to make them resilient to real context problems such as churn rate, node failure, performance degradation of the parent, etc, we propose variations of the same strategies, in which we impose a number of parents to each node. This adds more challenges since the overlay structure is more complex and it may result in peer interdependencies. These variations will be studied in Chapter 7 and will be designated with the suffix FP (fixed number of parents).

Chapter 7

Revisiting Peering Strategies in Push-Pull Based P2P Streaming Systems

7.1 Introduction

Studying peering strategies has been done in early works on P2P systems [SGMZ04, RS04] and more recent ones as well [RLC08]. Such systems had inherent limitations due to constraints imposed on their topologies (tree or multi-trees) or their content retrieval mechanism (pull). In the last case, the content retrieval is based on three steps at each hop: Sending buffer maps to neighbors, analyzing and requesting content and finally sending content. The additional delay resulting from such a heavy process is enough to impact the performance of good overlays and to make them struggle.

Thus, we believe that a push-pull scheduling mechanism [ZZSY07, LMSW07] is more suitable for comparing peering strategies in mesh overlays as it leads to much lower overheads and playback delays compared to the pull mechanism. In addition, it allows the proposal of new strategies that are based on overlay link delays: Such strategies were obsolete in pull-based system as the delay due to pull scheduling outweighs significantly the RTT (Round Trip Time) of an overlay link.

In Chapter 6, we have demonstrated that constructing a good overlay is a necessary step toward minimizing playback delay. We have proposed strategies that we compared, to state-of the art ones, using a push-pull scheduling algorithm. One of the strategies we propose, ***BP-MP***, (Best Parents, Minimum Parents) performed the best under some given scenarios and outperformed the strategy proposed in [RLC08] targeting at reducing playback delays in mesh overlays. However, the context of comparison was ideal: There were no overlay cycles and no dynamic selection of pushing nodes. As a consequence, after a start-up time, all streamed packets are being pushed.

In addition, the variations being compared consist in selecting the minimum number of parents until theoretically achieving the full streaming rate as done in [RLC08]. So, in these variations, a node may have a single parent or few parents. We designated variations with minimum number of parents by the suffix MP.

Having nodes with a small number of parents may be problematic in a real context because of the peer churn rate. For that reason, through simulations, the present chapter compares different variations of the same strategies, where each node has the same number of parents (when possible). Although such variations are more suitable to address churn, they result in complex mesh overlays. Some peer dependencies may occur which triggers the pull mechanism to come into action. In the remaining of the chapter, we designate variations with a specific number of parents by the suffix FP.

Thus, we propose two new variations ***BP-FP*** and ***PowerRTT-FP*** that we compare to two variations of the strategy proposed in [RLC08]: *Power-MP* and *Power-FP*. Results show that both ***BP-FP*** and ***PowerRTT-FP*** benefit better from the push-pull scheduling mechanism and lead to overlays with short source-to-end paths and thus to low playback delays.

This chapter is organized as follows. Section 7.2 presents the different strategy variations being compared. Simulation conditions are described in Section 7.3. In

Section 7.4, we present and comment the simulation results. We conclude in Section 7.5 and give hints on future work.

7.2 Peering Strategies

In the present section, we describe the variations of the peering strategies that we evaluate through simulations. These variations differ from the ones being compared in Chapter 6: Each node has a fixed number of parents instead of a minimum number of parents that provide him with the full streaming rate.

Therefore, we propose two new variations ***BP-FP*** and ***PowerRTT-FP*** that we compare to two variations of the strategy proposed in [RLC08]: *Power-MP* where a node selects the minimum number of parents (as in the original algorithm detailed in Section 6.3.4) and *Power-FP* where a node selects a fixed number of parents.

The comparison is based on the playback delay of peers. As we are addressing live video streaming, the playback delay is important especially if we want to move toward real time live video streaming where some sort of interaction can be possible among peers.

For each strategy, a joining node tries periodically to find new parents until reaching the number of needed parents. A node rejects a request for being a parent of another node if it has no more available upload slots. In that case the requesting node needs to look for a new parent.

We do not address the issue of the churn rate as our focus is on how to construct a good quality overlay. The algorithm of overlay construction can then be adapted to integrate some strategies to deal with churn.

Algorithm 7.4 BP-FP Selection

Input: L: List of eventual parents
Output: LP: List of parents (may be empty) {LP has a size between 0 and P (number of needed parents)}
Begin
$c \leftarrow current\_node$
sort nodes $p \in L$, in ascending order of the maximum distance of c through p: $p.maxDtoS() + \frac{RTT(p,c)}{2}$
while L is not empty and $Needed\_Slots > 0$ **do**
  $p \leftarrow Head(L)$
  $success \leftarrow RequestSlotsFromNode(p, 1)$
  if success **then**
    $Needed\_Slots \leftarrow Needed\_Slots - 1$
    $LP.insert(p)$
  end if
end while
End

7.2.1 The BP-FP Strategy

In the ***BP-FP*** strategy, a node tries to select P parents that provides it with the shortest paths to the source node. ***BP-FP*** is described in Algorithm 7.4: We evaluate the distance to the source according to the function $maxDtoS$ that we proposed in Algorithm 6.2. To reduce evaluating the distance indefinitely due to cycles in the overlay, a node will only update its distance relatively to a parent that has joined the session before it. Proposing the ***BP-FP*** strategy in a mesh overlay has been made possible thanks to the resource upload schema being used (see Section 6.2), where we drop the concept of neighbors and we use the concept of parents and children. Indeed, in neighbor based mesh systems, the symmetric relationship among nodes makes unsuitable such strategy.

7.2.2 The Power-FP Strategy

The *Power-FP* algorithm is a variation of The *Power-MP*: The only difference resides in the fact that in *Power-FP* each node has, when possible, a fixed number of parents,

P, while in *Power-MP* a node have at most P parents.

7.2.3 The PowerRtt-FP Strategy

The ***PowerRTT-FP*** algorithm is similar to the *Power-FP* algorithm except that it considers the RTT instead of the distance of the candidate node. Thus the *Power* of a potential parent p relatively to a node c is:

$$Power_{pc} = \frac{\min(AvailableSlots(p), P)}{rtt(p, c)}.$$

PowerRTT-FP has the merit to be simple and to consider the overlay link delay as an important criteria for parent selection. Although, ***BP-FP*** considers the RTT in the selection, its value is likely to be outweighed by the corresponding overlay path delay to the source.

7.3 Simulation

The objective of the conducted simulations is to measure the impact of the different compared peering strategies on the average playback delay observed by peers in the streaming session. The best strategy is the one that leads to the lower average playback delay. Through our proposed strategies, ***BP-FP*** and ***PowerRTT-FP***, we will show that taking informed and good peering decisions will result in a significant performance gain.

7.3.1 Simulator

We make use of the same simulator, node information and scenarios as in Section 6.4.1.

Push-Pull Algorithm

Like in Chapter 6, we are using the push-pull mechanism as already implemented in the simulator [ZZSY07](see Section 4.3 also).

This push-pull mechanism has a dynamic behavior that makes it able to adapt with more and less success to the overlay structure: If a streaming packet has not been pushed and has been pulled by another node then the subscription with the first node is replaced by a subscription with the new one. If the overlay contains complex dependencies between nodes, cycles for instance, there is no guarantee that all packets being received are pushed. Some of them will be pulled.

Before running the simulations, we introduced one modification in the push-pull algorithm which lies in the fact that we impose that the number of substreams being pushed by a node to be less or equal to the maximum number of its upload slots. This ensures that a node will not be overloaded.

7.4 Results

In this section, we present and comment the results obtained through the simulations. For each figure, the X coordinate axis designates the number of nodes (51, 101, 201, 401, 701 and 1001). Thanks to the fact that the push-pull scheduling mechanism restores the importance of an overlay link delay, we are able to come up with new strategies that do not necessarily have a big impact with a classical pull mechanism but recover all their power and relevance within the push-pull mechanism. The proposed strategies, ***BP-FP*** and ***PowerRTT-FP***, exploit this advantage to lead to low playback delays.

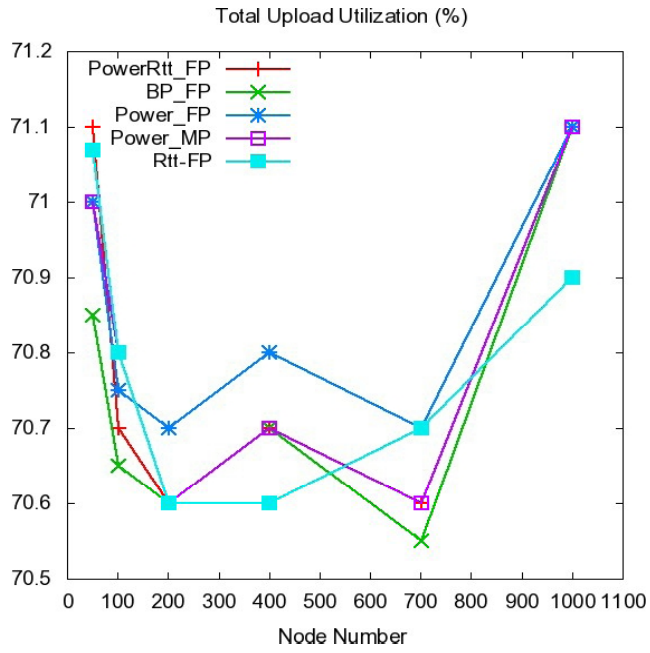


Figure 7.1: Total Upload Utilization - FP

7.4.1 Total Upload Utilization

The total upload utilization is the utilization ratio of the upload resource of all the P2P system, (see [ZZSY07]). The results of the total upload utilization of the P2P system are presented in Figure 7.1. The conclusion we draw is that the total upload utilization is almost the same for all compared strategies (maximum difference is about 0.3% or less for the same number of nodes). This is a very important result as it means that the compared strategies have almost the same efficiency with respect to resource utilization. Thus, we are able to state that the difference of performance of the compared strategies is not related to the upload resource utilization but rather to the quality of the strategy itself: If a strategy outperforms another one while using the same ressource, then the first one creates a better overlay. A similar result was obtained with MP variations in Chapter 6.

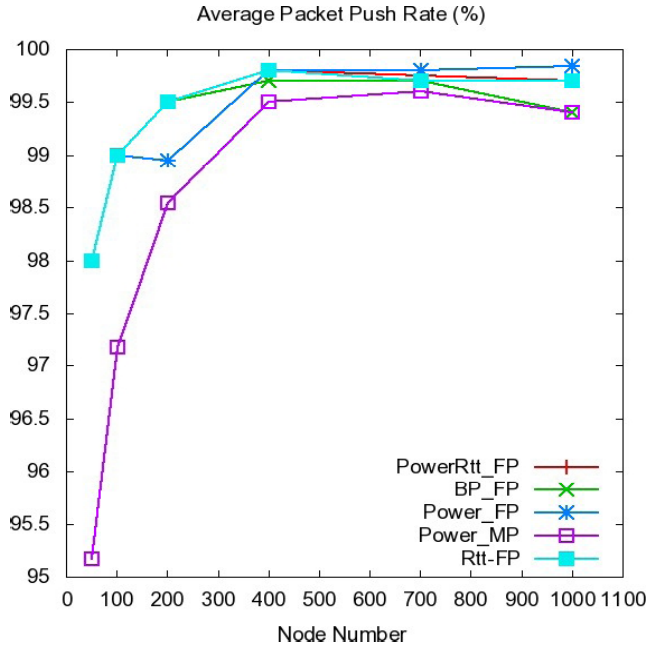


Figure 7.2: Average Push Rate - FP

7.4.2 Average Push Rate

The average push rate is the average over all nodes of the rate of packets that has been pushed (vs. pulled). The push rates of the different strategies are depicted in Figure 7.2. All the strategies led to a high pushed packet rate ($\geq 95\%$), for all scenarios. Thus the overhead has no significant value. We also note that starting from the scenario with 401 nodes all compared strategies have almost the same push rate: The difference is within 0.4%. Thus, any significant difference in the performance is not due to a difference of utilization of the pull mechanism but rather due to the quality of the overlay that has been constructed.

7.4.3 Average Playback Delay

We define the average playback delay as the average of playback delays of all nodes except the source. As in [ZZSY07], the playback delay depends on the ratio r of the received packets to the needed packets within buffer intervals of one second. Every

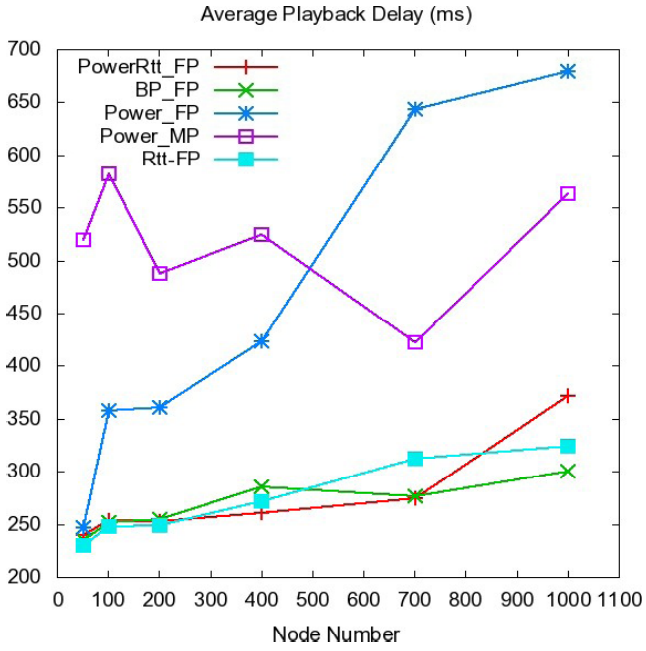


Figure 7.3: Average Playback Delay - FP

two consecutive intervals have a time slot lag of $250ms$. The playback delay is then determined by the lowest i^{th} time slot for which the corresponding buffer interval satisfies $r \geq 99\%$. The playback delay is then equal to $i \times 250ms$, i starting at 160 downward 0.

The results of the average playback delays of the peers are depicted in Figure 7.3. We note that the ***BP-FP*** and ***PowerRTT-FP*** strategies perform the best along with *RTT-FP*. ***BP-FP***, which is based on the maximum distance to the source, is successful at finding very short paths to the source. This has a good impact on the performance.

PowerRTT-FP, surprisingly, performs almost as best as ***BP-FP***. To analyze this result, we refer to Figure 7.4. We can see that ***PowerRTT-FP*** has usually the lowest average packet delay. Thus, this last strategy succeeds in building short paths to the source. *Power-MP* performs worse than ***PowerRTT-FP*** and ***BP-FP*** since:

- A node has the possibility to take the full content from one parent node while

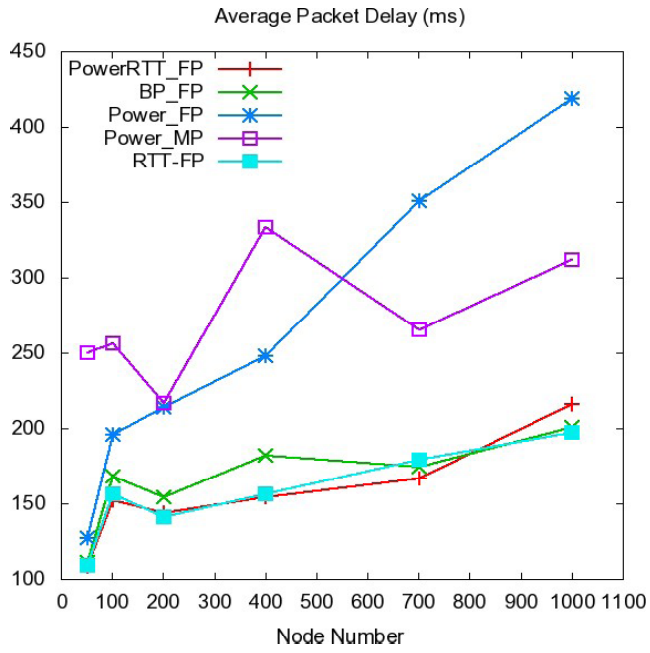


Figure 7.4: Average Packet Delay - FP

its upload bandwidth is lower than the streaming rate.

- In such strategy a node is sensitive to its parent performance. For instance, if a node is using the pull mechanism to get some part of the content it will experience a significant increase in its playback delay which will affect the playback delays of its descendants.

- A side effect may occur when calculating the fraction $Power_{pc}$ which may lead to bad peering decisions. As a consequence, we observe that the performance of this strategy is not stable: Zig-zags in the playback delay curve in Figure 7.3.

Power-FP performs relatively well with a low number of nodes. However, it leads to a serious performance degradation with a medium or a high number of nodes (≥ 401). Figure 7.4, shows that starting at 401 nodes *Power-FP* starts to have a high average packet delay meaning that the available overlay paths are long.

Compared to the ***BP-FP*** variation (see Figure 7.5), ***BP-MP*** leads to a better playback delays. This is due to the fact that it uses a minimum number of parents per

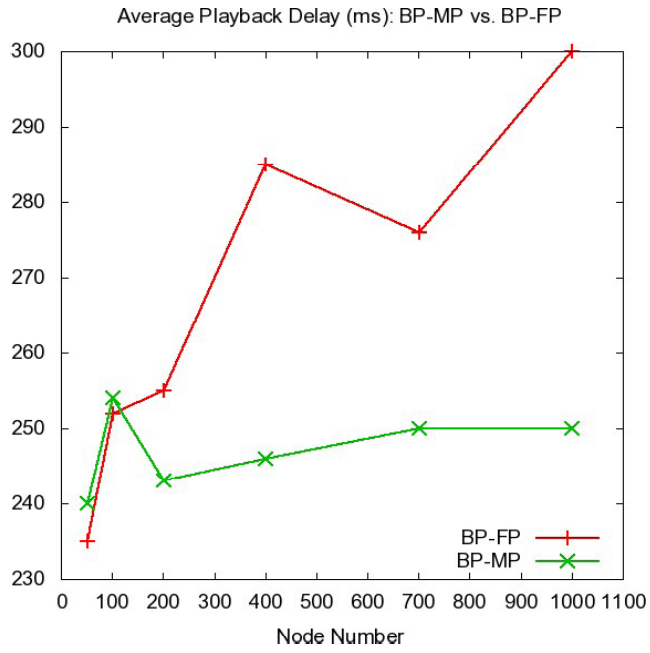


Figure 7.5: Average Playback Delay - *BP-FP*vs.*BP-MP*

peer and thus, it only introduces little discrepancy in the delivery delays experienced by a peer.

7.4.4 Average Packet Delay

The global average packet delay is obtained by computing the mean of average packet delay of all nodes except the source. It is calculated as in [ZZSY07]. Each node maintains its own average packet delay for packets that have been successfully received. The average packet delay at a node is a combination of the scheduling delay (due to the push and mainly pull mechanism) and of the overlay path length. When the pull mechanism is not used, all packets are being pushed, the average packet delay is almost equal to the average overlay path length.

Figure 7.4 shows that an average low packet delay is not enough to guarantee a low playback delay. The reason is that the packet delay deals with the successful reception of individual packets. Thus, it does not consider the contiguity of the

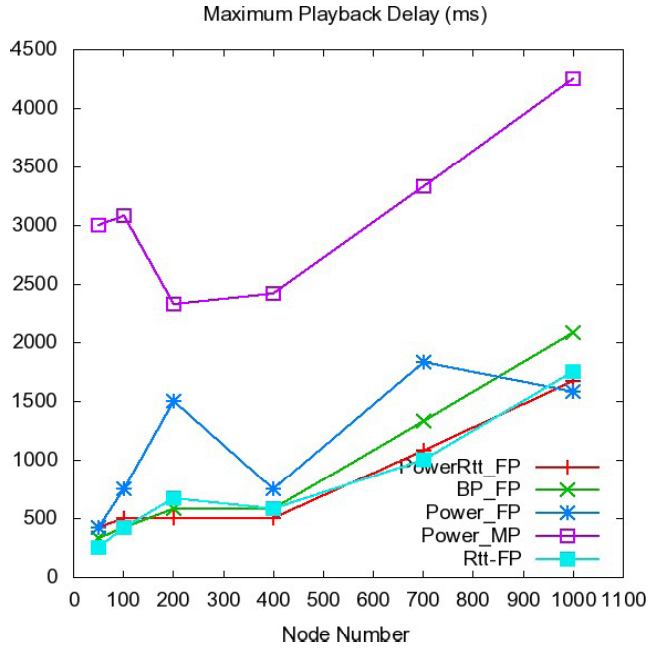


Figure 7.6: Maximum Playback Delay - FP

received packets while the playback delay does. For instance, in all scenarios but the one with 1001 nodes, although ***PowerRTT-FP*** has the best average packet delay, it is outperformed by the ***BP-FP*** strategy with respect to the average playback delay.

7.4.5 Maximum Playback Delay

The maximum playback delay results are presented in Figure 7.6. We observe that the ***BP-FP***, ***PowerRTT-FP*** and *RTT-FP* strategies perform generally better than the others. Despite the fact that it leads to an overall bad performance, *Power-FP* performs well with respect to the maximum experienced playback delay. This means that such a strategy leads to uneven playback delay distribution among nodes: Most of the nodes have a high playback delay comparatively to the other strategies. Since *Power-FP* has a high pushed packet rate, the main reason why this strategy leads to bad results resides in the fact that it constructs long overlay paths as already noticed

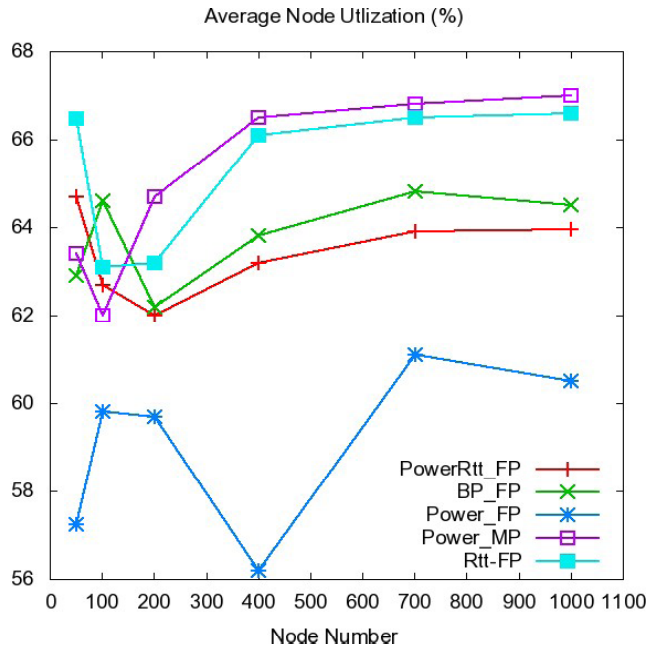


Figure 7.7: Node Upload Utilization - FP

through Figure 7.4.

Despite the fact that the *Power-MP* strategy has the worst maximum playback delay, it outperforms *Power-FP* for the scenarios with 701 and 1001 nodes with respect to the average playback delay. This means that *Power-MP* leads to a better playback delay distribution.

7.4.6 Average Node Upload Utilization

The global average node upload utilization is the mean of the average node utilization as computed in [ZZSY07]. The average node upload utilization results are presented in Figure 7.7. As the total upload bandwidth of the system is almost the same independently of the strategy being used (result obtained from Figure 7.1), the average node upload utilization becomes very interesting as it indicates how fair the load is divided among nodes. From Figure 7.7, we can see that *Power-MP* has the highest node upload load. This is due to the fact that when selecting a parent, a node tries

to reserve and use the maximum number of available upload slots until it reaches the number of upload slots needed to fully receive the streaming rate.

The ***BP-FP*** strategy usually has a high average node upload utilization which means that some nodes are experiencing high transmission loads comparatively to the others. This is due to the fact that the ***BP-FP*** strategy imposes high loads on the best nodes: A good node, i.e., with low distance to the source, has high probability to use all of its upload slots.

Power-FP and ***PowerRTT-FP*** strategies have the advantage of distributing the forwarding load more evenly over nodes since they take into account the available upload bandwidth of a potential parent at the moment of selection.

7.5 Conclusion

In this chapter, we have proposed two new peering strategy variations, ***PowerRTT-FP*** and ***BP-FP***, that are suitable for mesh based P2P streaming with a push-pull mechanism. With respect to playback delays experienced by peers, both strategies, especially ***BP-FP***, outperformed two variations of a recent peering strategy aiming at minimizing delays in P2P mesh systems. ***BP-FP*** has the best performance because it is based on a greedy approach where a node selects parents so that it has the shortest available paths to the source. This was confirmed by the fact that this strategy has usually the lowest average packet delivery delay.

When comparing the performance of ***BP-FP*** vs. ***BP-MP***, we note that the *MP* variation leads to lower playback delays because it uses less overlay links. Indeed, an *MP* variation selects the minimum number of needed parents and, subsequently, it will use a minimal number of links with the lowest delay.

In addition, and because *FP* variations leads to a more active parents per peer, it raises an additional issue which is the jitter caused by the discrepancy in the active

delivery path delays from the source to the concerned peer.

Subsequently, we have observed that a low average packet delay is not enough to guarantee that a strategy leads to a low playback delay since such a criteria does not consider the contiguity of received packets. We believe that having parents with too much different playback delays leads to a higher playback delay. Upcoming chapters, will partially investigate this issue and propose a pure push algorithm that reduces the impact of such a situation on the observed delays.

In order to validate the obtained results we propose a MILP model, through which we compare the overlay construction strategies independently from the push-pull scheduling strategy.

Chapter 8

A MILP Model for the Validation of Peering Strategies

8.1 Introduction

Although considered as a breakthrough technology not so long ago, P2P streaming has, so far, failed to break to the normal consumer who has been reduced to watch specific content, specially right protected materials and broadcasts, for free. Therefore, new initiatives are growing to allow P2P streaming systems to achieve their potential in the future Internet. This includes *P2P NEXT* and *NAPA WINE* from the European Future Internet Initiative [hi10]. In [ABE+09], there were discussions about P2P deployment issues related to digital rights protection and to the hostility of ISPs who try to minimize their inter traffic. Some suggestions were pointing that there must be a collaboration between P2P systems and ISPs about network topology and possibly hardware support through ISP based super peers.

The future Internet being seen as a network of applications more than a network of physical nodes, we strongly believe that P2P streaming systems should also focus on the unexplored perspectives that they offer and that are impossible with classical systems. For instance, besides the video delivery, we can easily build many other

services such as a social network where *friend peers* interact and react to the content being displayed. Therefore, based on the advantages that derive from the use of a P2P architecture such as the resource scalability, we still believe that a P2P streaming system has its place in the jungle of content distribution providing that we find the right way to evaluate and to guarantee its performance.

Due to many random factors, such as the node join distribution and characteristics, it is difficult to evaluate a P2P system. Each system has its own combination of a peering strategy and a scheduling mechanism. The evaluation depends on the implementation of each part, including a set of assumptions and parameter tuning procedures.

To help addressing this issue, we propose a Mixed Integer Linear Programming (MILP) model that allows us to evaluate a peering strategy independently from the scheduling mechanism. Based on our previous works [OKJ09b, OKJ09a], we claim that new peering strategies should be proposed with the objective of collecting and taking advantage of information about the overlay, such as the distance of a node to the source. Such strategies were found to construct better overlays that help decreasing the playback delay of peers.

However, the results presented in Chapter 6, were obtained in an ideal situation (all streaming packets being pushed) while results in Chapter 7 were dependent on the scheduling algorithm. Resolving these issues constitutes the scope of the current chapter. We validate the results obtained in Chapter 7, by solving the MILP model. With respect to a specific objective, this model constructs an optimal push scheduling strategy and thus the playback delay will depend solely on the overlay quality.

This chapter is organized as follows. Section 8.2 describes the different algorithms being compared. In Section 8.3 we validate the simulation results obtained in Chapter 7 through a MILP model. We conclude in Section 8.4 and give hints on future work.

8.2 Peering Strategies

In this section, we describe the peering strategies being compared. Through previous Chapters 6 and 7, we showed that POS strategies combined with push-pull scheduling have the potential to outperform BS strategies with respect to playback delay. We also proposed a new peering strategy ***Best Parents, (BP)***. It is a POS strategy as it exploits information about the maximum distance of an end node to the source to make peering decisions. We compare it to a well known BS strategy, *RTT* or *Closest Parents*, where a node selects the nearest parents based on the RTT, and to a POS strategy *Power* proposed by Ren *et al.* [RLC08]. This led us to propose another new POS strategy, ***PowerRTT***, that has a hybrid selection criterion inspired by *RTT* and *Power* strategies.

We compare these strategies using a slot based resource schema (see Section 6.2 for more details): We divide the video stream into SR substreams and the upload bandwidth of a node into slots. Each slot size is obtained by multiplying the substream size by a parameter $0 < \alpha < 1$. α accounts for overhead and retransmissions.

In Section 6.5 and [OKJ09b], we dealt with *Minimum Parents* (MP) variation of the strategies: A peer requests all the available upload bandwidth of a potential parent until totalizing the streaming rate. In Section 7.4 and [OKJ09a], we dealt with *Fixed Parents* (FP) variations: we impose the same number of parents $P = SR$ to each node.

8.3 FP Result Validation

As mentioned in Chapter 7, the performance of a P2P session depends on some random decisions taken to map substreams to be pushed to corresponding parents. For that reason we had to simulate each streaming session several times and then

take the average values of the results. Thus, the performance depends strongly on the content retrieval mechanism being used.

In this section, we propose a validation process that allows the comparison of the overlays being constructed by the different FP algorithms with an optimized retrieval mechanism. Subsequently, the overlay leading to the best results is considered to be constructed by the best algorithm.

8.3.1 Validation Process

The validation process we propose is based on two steps. Firstly, using the simulator, we construct the overlay according to the selected strategy. Secondly, the constructed overlay represents the input of a MILP model that outputs an optimal rate assignment for the overlay links. The overlays being used are the ones obtained in Section 7.4.We use IBM ILOG CPLEX [CPL10] to solve the proposed MILP model. This process is repeated three times for each strategy/node scenario. The best strategy is the one that leads to the lowest average playback delay.

8.3.2 MILP for the FP Validation problem

We propose a MILP model that proceeds by constructing a distribution tree (spanning tree) for each substream. Solving the proposed model provides us with the transmission rates over the overlay links. The transmission rate distribution is optimal with respect to the selected objective function and is equivalent to a pure push scheduling mechanism.

Notations and Definitions

Let V be the set of peers in the network. Let P_v be the parents of a peer v. S is the streaming source. We have $S \in V$ and P_S is empty. Denote by $V^\star$ the set of nodes

deprived from the source node. Let SR be the streaming rate in slot unit. In our specific case, we selected $SR = 5$ as it is the maximum streaming rate that can be handled by the considered P2P system.

Parameters and Variables

As mentioned before, the overlay links are obtained from the result of the simulation. Let $e \in V \times V$ be a state vector such that $e_{uv} = 1$ if there is an overlay link from node u (parent) to node v (one child of u), $u, v \in V$, $u \neq v$ and $v \neq S$, i.e., u may transmit data directly to v. We denote by b_v^{UL} the maximum upload bandwidth of node v.

Let d_{uv}^N be the end-to-end latency, at the physical network level, of the node v relatively to the node u for $u, v \in V, u \neq v$.

We have $2 \times SR$ different vectors: SR rate transmission vectors $(r^i, 1 \leq i \leq SR)$, and SR delay vectors $d^{A,i}$, which are described next.

r^i is the transmission rate vector relatively to substream i, where each component r_{uv}^i denotes the transmission rate from node u to node v, for $u \neq v$, with respect to the substream i.

r^i must satisfy the following relations:

- if $e_{uv} = 0$ then $r_{uv}^i = 0$, $1 \leq i \leq SR$.
- if $e_{uv} = 1$ then $r_{uv}^i = 0$ or $0.99 \leq r_{uv}^i \leq 1$, $1 \leq i \leq SR$.

Vectors $d^{A,i}$ correspond to delivery delays, i.e., $d_v^{A,i}$ is equal to the delivery delay, from the source S to a node $v \in V^\star$, of a packet that belongs to substream i. For the source node S we have:

$$d^{A,i}(S) = 0, \qquad 1 \leq i \leq SR.$$

Optimization Criterion

We propose to consider an objective with two characteristics in mind. First, the objective function must be simple: As we are dealing with a MILP problem that may be hard to solve, we need a function that is easy to optimize. Second, the objective function must help toward minimizing the playback delay experienced by the nodes.

Thus, we propose the following objective:

$$f^{\text{OBJ}} = \frac{\sum_{v \in V^\star} \sum_{u \in V} \sum_{1 \leq i \leq SR} r_{uv}^i \times d_{uv}^N}{(|V| - 1) \times SR}.$$

The proposed objective minimizes the delay weight of the delivery trees that are being constructed. It can be seen as a minimization of the global delivery cost, in terms of delay, of one slot of the video content. It is simple because the only variables being used are semi-continuous ones with small range (equal to 0 or to a value between 0.99 and 1). It helps minimizing the playback delay because it favors links with low latencies.

The obtained solutions are within 1% of the optimal solution.

Constraints

Delay Constraints. They express upper (14) and lower (15) bounds of the delay experienced by a node in one of the trees. They help minimizing the playback delays by enforcing the smallest possible values.

$$d_v^{A,i} \geq d_p^{A,i} + d_{pv}^N - M(1 - r_{pv}^i) \qquad v, p \in V, p \neq v, v \neq S, 1 \leq i \leq SR \qquad (14)$$

$$d_v^{A,i} \leq 1 + d_p^{A,i} + d_{pv}^N + M(1 - r_{pv}^i) \qquad v, p \in V, p \neq v, v \neq S, 1 \leq i \leq SR. \qquad (15)$$

The constraints (14) and (15) use M, a constant such that constraints where M

appears become redundant if $r^i_{p,v} = 0$.

Rate vs. Bandwidth Constraints. The first set of constraints states that a node $v \in V$ can not upload more than its maximum capacity:

$$\sum_{c \in V^\star \setminus \{v\}} \sum_{1 \leq i \leq SR} r^i_{vc} \leq b^{\text{UL}}_v, \qquad v \in V. \tag{16}$$

The next set of constraints states that each node $v \in V^\star$ is receiving enough data to handle each video substream i:

$$\sum_{p \in V \setminus \{v\}} r^i_{pv} \geq 0.99, \qquad v \in V^\star, 1 \leq i \leq SR. \tag{17}$$

The last set expresses conditions on the transmission rate from a parent p to a child node v.

$$r^i_{uv} \leq e_{uv} \qquad u, v \in V^\star, u \neq v, 1 \leq i \leq SR. \tag{18}$$

Solution Constraints. Their goal is to impose some characteristics on the obtained solution. The first set expresses that a peer can send no more than $MaxOccurence$ substreams to a given child:

$$\sum_{1 \leq i \leq SR} r^i_{vc} \leq MaxOccurence, \qquad v \in V, c \in V^*, MaxOccurence \geq 1. \tag{19}$$

In our experiments, we considered the worst scenario as it leads to the highest delay cost. Hence, we have $MaxOccurence = 1$: A client peer can send no more than one substream to a child. This is the worst case because the more links we use, the more the delivery delay increases.

To reduce the solution space, the second set imposes an upper bound, var, on the

difference among the average delivery delays in the trees:

$$\max_{1 \leq i \leq SR} \frac{\sum_{v \in V^*} d_v^{A,i}}{|V^*|} \leq \min_{1 \leq i \leq SR} \frac{\sum_{v \in V^*} d_v^{A,i}}{|V^*|} + var \tag{20}$$

We set var to the time interval between two consecutive packets in the video stream, i.e.,

$$var = \frac{1000}{SR} ms.$$

Let us assume that the first packet is created and sent at time $t = 0$. The average time at which a packet will be received by each node in the corresponding i_{th} tree is

$$\frac{1000}{SR} * (i-1) + \frac{\sum_{v \in V^*} d_v^{A,i}}{|V^*|}, 1 \leq i \leq SR$$

The time lag between receiving a packet of the first substream (assumed to be $d^{A,1}$) and a consecutive packet of the last substream (assumed to be $d^{A,SR}$) is:

$$(SR-1) * \frac{1000}{SR} + d^{A,SR} - d^{A,1}$$

If we impose such lag to be less than $1000ms$, we set var, in our specific case, to $200ms$. Experimentally, this value leads to good results without impacting the execution time.

MILP Model Flexibility

The proposed MILP model has many advantages. First, it is independent from the peering strategies. If we obtain a solution, it means that we have enough distribution trees. If there is no solution, it means that there is no push scheduling possible for the given overlay.

Additionally, the proposed model can be seen as scalable enough. Indeed, we have obtained solutions for overlays with 1001 nodes and 5000 links in reasonable times.

Reducing the streaming rate to $SR = 4$, for instance, will allow us to increase the number of nodes. In a future work, we will analyze the model in more depth and try to provide more solid evaluation of its scalability.

From a P2P designer perspective, it is possible to extend the model to analyze the performance of design choices. For instance, we may want to construct $SR + 1$ trees instead of SR trees. The additional tree is used for protection purpose in case of node failure/departure. The proposed model will help analyzing the impact of such an additional tree on the performance.

From an ISP perspective, the model can easily be extended with constraints to bound RTT links between nodes, and thus reduce inter-ISP traffic or with constraints that bound traffic volumes between nodes with high RTT (likely not belonging to the same ISP).

8.3.3 Results

The first observation that we obtain from the results is that all overlays lead to solutions with the MILP model, i.e., it was possible to construct a spanning tree for each substream. Thus, we conclude that the push-pull scheduling strategy, which has been used in the simulations in Chapter 7, was unable to construct these distribution trees. If that was the case, all the packets would have been pushed and the pull scheduling would not have been triggered.

Average Playback Delay ($AvgPlayD$)

The playback delay of a node $v \in V^\star$, $PlayD(v)$, is defined as the maximum delivery delay experienced by a node to receive a substream:

$$PlayD(v) = \max_{1 \leq i \leq SR} d_v^{A,i}.$$

With the assumption that overlay link delays remain unchanged, $PlayD(v)$ is the lowest time at which v may start playing the received video stream without taking a risk of running out of packets. Then we get

$$AvgPlayD = \frac{\sum_{v \in V^\star} PlayD(v)}{|V^\star|}.$$

The results of the average playback delay are presented in Figure 8.1. The ***BP-FP*** strategy generally performs the best compared to the FP strategies. This is a confirmation of the results obtained by simulation in Section 7.4. So we can state that the ***BP-FP*** strategy constructs the best overlay, within compared strategies, with respect to end-to-source distance which leads to lower playback delays than the other strategies.

The *Power-MP* strategy performs best with a low number of nodes (51 to 201). For the other scenarios, it leads to an average playback delay that is 10% to 21% higher than ***BP-FP***. This is due to the fact that the average packet delay (see section 8.3.3) of *Power-MP* is significantly higher (25% to 49%) than the one obtained with ***BP-FP*** as it can be seen in Figure 8.1.

The *Power-FP* strategy performs the worst as it usually has the worst average packet delay. ***PowerRtt-FP*** and *RTT-FP* perform generally very well. ***PowerRtt-FP*** leads to average playback delays that are 1.15% to 21%, higher than the ones of ***BP-FP*** while *RTT-FP* leads to average playback delays that are, -4.7% to 26.83%, higher than ***BP-FP***.

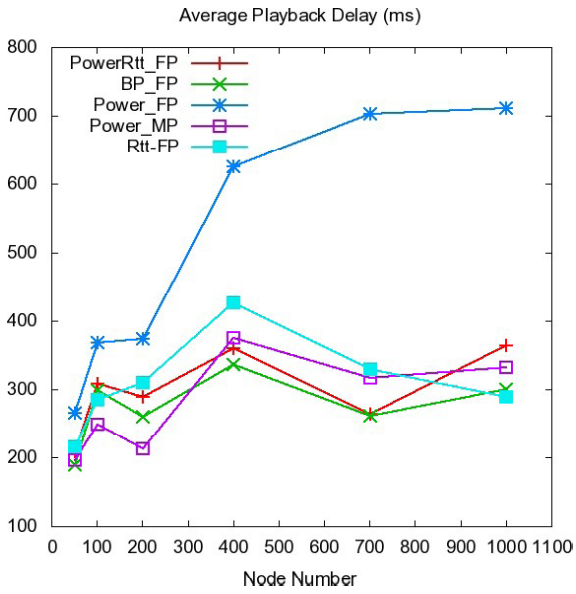


Figure 8.1: Average Playback Delay - Validation

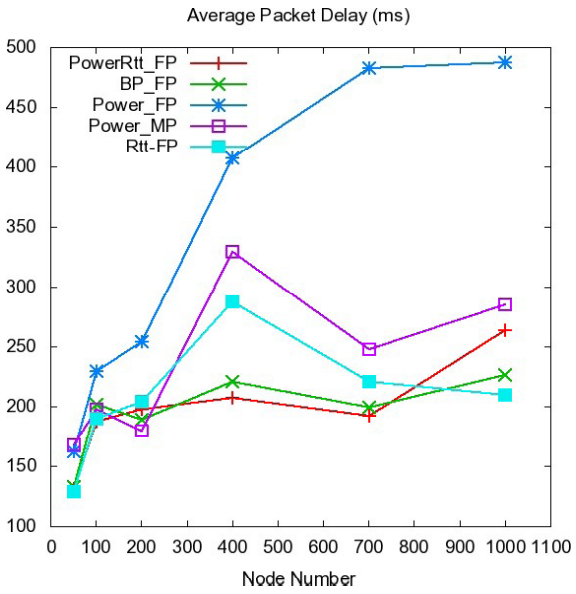


Figure 8.2: Average Packet Delay - Validation

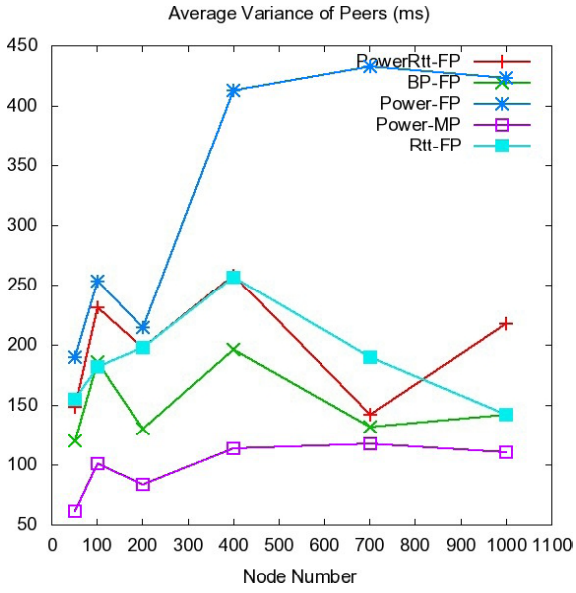


Figure 8.3: Average Variance - Validation

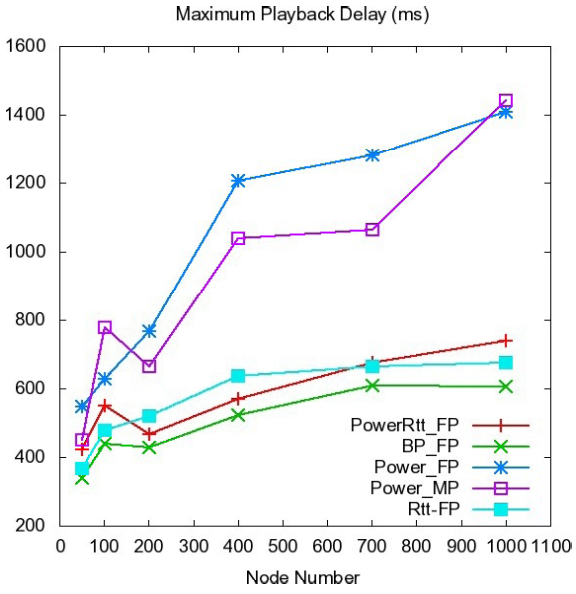


Figure 8.4: Maximum Playback Delay - Validation

Average Packet Delay ($AvgPackD$)

The average packet delay is defined as the average delivery delay of all packets to all client nodes:

$$AvgPackD = \frac{\sum_{1 \leq i \leq SR} \sum_{v \in V^\star} d_v^{A,i}}{|V^\star| \times SR}$$

Figure 8.2 depicts the average packet delays obtained by the compared strategies. ***PowerRtt-FP*** and ***BP-FP*** have usually the lowest $AvgPackD$ for each scenario. Figure 8.2 also shows that having a low average packet delay is not sufficient to guarantee a low playback delay. For instance, in the scenario with 401 nodes, the *RTT-FP* strategy has a lower $AvgPackD$ than *Power-MP*. However, the latter has a better playback delay. We observe a similar situation in the scenario with 1001 nodes where the ***PowerRtt-FP*** has a better $AvgPackD$ than *Power-MP* while having a worse $AvgPlayD$.

To analyze these observations, we use Figure 8.3 which depicts the average end-to-source distance variance of the client nodes. We define the average maximum variance of a node as the difference in delivery delay between the latest received substream and the earliest received one :

$$MaxVar(v) = \max_{1 \leq i \leq SR} d^{A,i}(v) - \min_{1 \leq i \leq SR} d^{A,i}(v), v \in V^\star.$$

The variance can be seen as a jitter at the overlay level. Then, we obtain:

$$AvgMaxVar = \frac{\sum_{v \in V^\star} MaxVar(v)}{|V^\star|}.$$

Thanks to Figure 8.3, we observe that the degradation of performance of ***PowerRtt-FP*** with 1001 nodes, and *RTT-FP* with 401 nodes, comes with a dramatic increase in the average variance.

Actually, an increase in the average variance has a good probability to result in an increase in the playback delay. Indeed, the latter is computed as the maximum of the delivery delays throughout the distribution trees. We conclude that a low playback delay is the result of the combination of a low average packet delay with a low average variance. This is confirmed, for instance for the ***BP-FP*** strategy as it usually has the lowest average packet delay and the lowest average variance, as observed in Figure 8.2 and Figure 8.3. This also means that it did not suffer much from the increase of the parent degree i.e., comparatively to its MP variation.

Power-MP has the lowest average variance (thanks to a reduced parent degree), while having the second worst average packet delay. The low variance has shortened the impact of the high packet delay to lead to a good playback delay performance (usually within the best three, see Figure 8.1). This strategy suffers, however, from an increase in parent degree as *Power-FP* has the worst performance in every aspect.

Maximum Playback Delay ($MaxPlayD$)

We define the maximum playback delay as:

$$MaxPlayD = \max_{v \in V^\star} PlayD(v).$$

The maximum playback delay results are depicted in Figure 8.4. We can see that the ***BP-FP*** strategy leads to the lowest $MaxPlayD$'s.

Although *Power-MP* has high $MaxPlayD$'s, it performs well with respect to the average playback delay. This observation stands even when *Power-MP* performs the best (with 101 and 201 nodes). This means that the playback distribution is not evenly distributed: some nodes have high playback delays while others have low delays.

8.4 Conclusion

In the present work, we have compared and analyzed several peering strategies. We have investigated two types of strategies. The first ones, Basic Strategies (BS), use local or relative information about peers while the second ones, Partially Overlay-aware Strategies (POS), exploit some information on how a node is doing in the overlay such as its end to source delay.

We have proposed the *Best Parents* POS strategy, ***BP***, that has proved to be the best in all considered scenarios. Thus, the informed peering decisions had a positive impact on the performance. POS strategies benefit greatly from the utilization of the push-pull scheduling mechanism, as the overlay link delay is no more outweighed by the pull scheduling delay.

Through the simulations and the result validation, we have showed that an overlay with a low global average packet delay is not a sufficient condition to lead to an average low playback delay. We have identified the *Variance* concept, difference in source to end path distances to the same end node, as a dominant criteria for the performance. A low playback-delay is a combination of a low packet delay and a low variance.

To validate results, we have proposed a new process using a mixed integer linear programming model. This process can be used to validate the delay performance of any overlay created following a specific strategy. It is equivalent to having a solution using an optimal push scheduling approach. Using the push approach leaves the overlay quality as the main criteria that impacts the playback delay performance.

Unlike simulations and experimentations, the proposed evaluation process is independent from the scheduling strategy or implementation choices and thus gives a more stable and authentic evaluation of a peering strategy. The proposed model can be extended to suit the view of a P2P designer or an ISP and, therefore, to study the impact of design decision such as locality awareness based on the RTT or protection

against churn.

The performance of a P2P system is still dependent on random events such as the node join distribution. Addressing such an issue is a big challenge but is worth investigating because, when combined with our evaluation process, we will have a P2P system that has a near deterministic behavior.

We have also seen that overlays with a high parent degree are complex and may contain cyclic dependencies that lead to the utilization of the pull mechanism and, consequently, to a playback delay increase. The used push-pull algorithm did not manage to get rid of the interdependencies. Thus, we plan to propose a new pure push scheduling strategy as it seems to be the best solution. Although some recent works exist on such an approach, it is still challenging to deploy it on mesh overlay networks.

Part III

Content Retrieval

Chapter 9

Push Scheduling and Mesh Overlays: The Best of Both Worlds

Due to their multi-parent ability, mesh based P2P systems need an elaborated scheduling strategy to retrieve content from different peers and to avoid redundancy. One common approach is based on the pull mechanism where a peer requests content based on the buffer maps of its neighbors/parents. Alternate approaches have been proposed such as push-pull and more recently some push strategies. Unfortunately, although push-pull strategies improve the playback delays compared to pull ones, they do not perform well at startup and at handling the recovery of lost packets because they still rely on the pull mechanism. The few existing push strategies follow complex algorithms and still rely on the exchange of buffer map information among nodes to avoid redundancy.

In this chapter, we propose a new pure push strategy, *PurePush*. *PurePush* is derived from a push-pull real world mechanism developed by Zhang *et al.* (2005), in which we eliminate the pull part and replace it by a probabilistic push. We next propose two variations, *BPP* and *APP*, which both outperform the initial push-pull strategy with respect to playback delay in the realistic case where we consider packet

losses. Moreover, *APP* leads to the best result while having the lowest overhead and redundancy.

9.1 Introduction

Peer-to-Peer (P2P) architectures are considered as attractive and scalable solutions for video streaming. They do not require Internet infrastructure modifications and they help eliminating bandwidth bottlenecks at the content source server. The performance of a P2P system depends on the overlay that it constructs. For instance, while tree-based P2P systems have an inherent push scheduling strategy but a weak resiliency, mesh-based systems need more elaborated scheduling strategies. Such strategies need to deal with the complexity of mesh overlays and to avoid redundancy due to the multi-parent ability. Therefore, they result in high playback delays.

Some recent works addressing playback delays, propose push-pull mechanisms [ZZSY07, LMSW07] or improved scheduling strategies [Liu07, CXH08]. Although few push strategies have been proposed, they rely on a heavy scheduling of substreams to avoid data redundancy [BLPL$^+$08]. Some other works use random network coding to resolve the redundancy issue [WL07] but seems to be efficient with low streaming rates only ($64kbps$). Another common approach is to utilize gossip/epidemic push strategies where either receiving nodes or packets to be sent are picked randomly or according to certain criteria [BMM$^+$08]. All the above solutions did not make it through to real P2P applications. In addition, the delays resulting from scheduling and from recovering lost packets are a big obstacle toward bringing more interactive applications to P2P content delivery systems.

The authors of [ZSXY08] propose to guarantee delay for P2P live streaming over Internet through the utilization of a push-pull based mechanism within a static environment. The key idea is that, in order to meet the delay requirements, the source

doubles or reduces the number of times a packet is sent directly to peers without explicit requests. In other words, the server is responsible for covering bad peering and scheduling decisions by increasing its upload transfer volume in order to send more packet copies directly to other peers.

Scheduling strategies have a critical role in reducing the playback delays through reducing the scheduling delay. Thus, in the present work, we propose a new scheduling strategy for live video streaming, called *PurePush*, which combines the best of both worlds: Mesh overlays known for their resiliency, and push scheduling strategies known for their low delays. Our starting point is the real world push-pull algorithm of [ZZSY07]. The *PurePush* design eliminates completely the pull mechanism and replaces it by a probabilistic push. Therefore, it leads to a fast delivery of packets and to a fast recovery of lost packets. This makes it suitable for interactive but non critical applications as it is a best effort solution.

PurePush acts like a multi-tree when packets are not lost as only parents in the distribution trees (over the mesh overlay) are involved while it acts like a mesh in the case of packet loss as all parents are involved in the recovery process. *PurePush* does not require any buffer map exchange, no special video encoding and is simple to implement.

Although *PurePush* has some similarities with gossip protocols, it is different in the sense that:

- A peer does not select the packet to be sent randomly. It randomly forwards the last received packet to its neighbors: All or some neighbors will receive the packet while the others will not.
- The probabilistic forwarding of packets is used only at startup and to recover lost packets.
- A peer tries to assign the task of forwarding a video substream to a specific

parent. Non lost packets that belong to the assigned substream will be relayed by that parent as long as it is able to do it.

We consider two variations of *PurePush*: *Basic PurePush*, a straightforward implementation called *BPP*, and, *Advanced PurePush*, an improvement of the *BPP* strategy called *APP*. We proposed an earlier version of *BPP* in [OKJ11]. However, it was evaluated in a different context than in the present work: The packet loss rate was equal to 1% under the assumption that a streaming packet cannot be lost twice between the same peers. The current chapter presents a more detailed desccription and performance evaluation of the *BPP* variation specially with respect to the overhead and the redundancy. In addition, we consider higher packet loss rate (2%) under the assumption that a packet can be lost more than once on the same overlay link.

Compared to *BPP*, *APP* integrates more intelligence in the choice of potential pushers at startup and following a request of lost packets. It leads to a very low redundancy and overhead, even less than the push-pull algorithm of GridMedia (see Section 4.3).

Simulation results show that *PurePush* outperforms by far the push-pull algorithm of [ZZSY07] in a packet loss context. In particular, *APP* succeeds in fulfilling the potential of combining the low scheduling delay of push strategy with the resiliency of mesh overlays.

This chapter is organized as follows. In Section 9.2, we recall the push-pull algorithm of [ZZSY07] that represents our starting point. Section 9.3 presents the pure push approach we propose. Simulation conditions are described in Section 9.4. In Section 9.5, we present and comment the simulation results. We conclude in Section 9.6 and give hints on future work.

9.2 Push-Pull Algorithm

9.2.1 The Original Push-Pull Mechanism

In the present work, the starting point is the push-pull mechanism proposed by Zhang *et al.* [ZZSY07] and that we designate by *PushPull* (See Section 4.3). In *PushPull*, the stream is divided into n substreams according to the sequence number of packets. Contiguous packets from different substreams form a packet group. k contiguous groups form a packet party. Each group has a group ID from 0 to $k-1$.

The group with ID equal to 0 is chosen as the base to take push scheduling decisions, i.e., requesting a push subscription from one peer or canceling an existing subscription. The pusher of a substream is the parent from which the node has successfully pulled a packet that belongs to both the wanted substream and a group with ID 0. This way, the switching among pushers does not occur too frequently. Other received packets that do not belong to group 0 do not impact push subscriptions.

At startup, a node requests packets using the pull mechanism with random scheduling until receiving packets from specific substreams. Then, it subscribes to senders so that each sender relays the packets of subscribed substreams directly to it. If the receiving quality of a node is greater than 95%, it does not request buffer maps anymore.

Lost packets are recovered the same way. If a streaming packet has not been pushed and has been pulled by another node then, the current subscription with the first node is replaced by a subscription with the latter one, (always at the beginning of a group with ID 0).

This dynamic behavior allows the algorithm to adapt with limited success to the overlay structure. If the overlay contains complex dependencies among nodes such as, e.g., cycles, there is no guarantee that all packets being received are pushed. Some

of them will be pulled.

The content distribution does not necessarily follow exactly the constructed overlay. A node may have 5 parents but, for instance, only 2 are sending the full content.

9.2.2 Modifications

We introduced two main modifications to improve the original algorithm. Firstly, as we divide the bandwidth into slots (see Section 6.2), we impose that the number of substreams being pushed by a node be lower or equal to the maximum number of its upload slots. This ensures that a node will not be overloaded. Secondly, there is a periodic monitoring of pushers. If, at node c, there is no pusher for a given substream, then c eliminates its worst parent, i.e., the parent with the lowest transfer volume to c, provided it is less than a substream size. Then, c looks for a new parent even if there is no elimination. Indeed, if each current parent is providing at least one substream to c then, c has not enough parents and thus, has to look for an additional one.

The last modification enhances the dynamic adaptation of the algorithm to the overlay topology. Indeed, in Chapter 7, simulations in an environment without packet loss have shown that not all packets were pushed. Some of them were pulled. We attributed this result to the complexity of the overlay and to the interdependencies among nodes that the algorithm failed to deal with.

9.3 Pure Push

As long as the pull part in a push-pull scheduling strategy is not triggered into action, it is possible to achieve excellent results with respect to throughput and playback delays. Thus, in this section, we propose a pure push approach that takes, as a starting

point, the enhanced push-pull algorithm of the last section and then eliminates the pull part. As already observed, the pull part was mainly used at the startup when new nodes join the session and to recover lost packets. We replace it by a probabilistic push in both situations. We also eliminate all exchanges of buffer maps at any time during the session.

Algorithm 9.5 Relaying a Packet in *BPP*

Input:
p: Current node
$ChildInfo$: Array of children information of p
$SeqNum$: Sequence number of the packet to be relayed
RP_{BPP}: Relay probability equal to $1/NumberOfParents$
RND : Integer random number between 1 and 100
Begin
$SubStream \leftarrow SeqNum$ mod $NumberOfSubstreams$
for each child c of p **do**
  $c.Child\_Missing\_Pusher \leftarrow (ChildInfo(c).PushersMap$ is empty **or**
  $ChildInfo(c).PushersMap[SubStream]$ is unknown)
  if $c.Child\_Missing\_Pusher$ is *true* **then**
    if $RND \leq 100 \times RP_{BPP}$ **then**
      Relay packet $SeqNum$ to c
    else
      Do not relay packet
    end if
  end if
end for
End

9.3.1 Start up and substream scheduling

At startup, an overlay network is constructed using the peering ***BP-FP*** strategy variation we proposed in Chapter 7 including the modifications suggested in Section 9.4.2. A node looks for P parents such as $P = S$ where S is the number of substreams of the video. When a node p receives a request in order to be a parent of a node c, it checks if it has any available upload slots. If not, it sends back a denial message to c.

Algorithm 9.6 Relaying a Packet in APP

Input: p: Current node
$ChildInfo$: Array of children information of p
$SeqNum$: Sequence Number of The packet to be relayed
RP_{BPP}: Maximum Relay probability equal to $1/NumberOfParents$
RND: Randomly generated integer between 1 and 100
Begin
$SubStream \leftarrow SeqNum$ mod $NumberOfSubstreams$
for each c child of p **do**
  $c.Child\_Missing\_Pusher \leftarrow (ChildInfo(c).PushersMap$ is empty **or**
  $ChildInfo(c).PushersMap[SubStream]$ is unknown)
  if $c.Child\_Missing\_Pusher$ is $true$ **then**
    if $p.PushersMap[[SubStream]]$is known && $p.AvaliableUploadSlots > 0$ **then**

      $RP_{APP} = \min(\ RP_{BPP}/(\frac{1000+\beta \times rtt(p,c)}{1000}), \frac{p.AvailableUploadSlots}{p.MaximumUploadSlots})$
      {β: Integer to amplify the RTT value}

    else
      $RP_{APP} = \frac{RP_{BPP}}{NumberOfParents}$
    end if
  end if
  if $RND \leq 100 \times RP_{APP}$ **then**
    Relay packet $SeqNum$ to c
  else
    Do not relay packet
  end if
end for
End

Algorithm 9.7 Detecting packet loss and Request Process in *BPP*

Input:
c: Current node
$SeqNum$: Sequence number of the received packet
$LastReceived[i]$: Sequence number of the last received packet of the substream i
$MaxMissing$: Maximum number of packets that can be recovered
Begin {This processing is executed at the reception of each packet}
$SubStream \leftarrow SeqNum \bmod NumberOfSubstreams$
if $SeqNum - LastReceived[SubStream] > NumberOfSubstreams$ **then**

$MissingNumber = \min(MaxMissing, \frac{SeqNum - LastReceived[SubStream]}{NumberOfSubstreams} - 1)$

create a Request Packet Array $ReqPackFromParents$
for $i = 0$ **to** $MissingNumber - 1$ **do**
for each parent p **do**
Insert packet$(SeqNum - (i+1) \times NumberOfSubstreams)$ into $ReqPackFromParent[p]$
end for
end for
end if
End {The request period is fixed in *BPP*}

Otherwise, it adds c to its list of children. Additionally, each node c has a push map. i.e., an array of S bits indicating whether c has a pusher or not for each substream. The size of a push map is far less than the size of buffer maps that are used in most existing systems. When the push map of a node changes, it is sent to all parents. Thus, we expect to have a low overhead.

Parent Side: Starting from the time when p accepts the request to be a parent of c, every packet received by p is eligible to be pushed to c. First, p determines the substream of the received packet. Then, if c has already subscribed to this substream with p, the packet is pushed to c. Otherwise, p needs to know whether c is missing a pusher for the concerned substream or whether c has subscribed to this substream with another parent. If c is missing a pusher, the packet is relayed. These steps are detailed in Algorithm 9.5 for *BPP* and Algorithm 9.6 for *APP*.

Algorithm 9.8 Detecting packet loss and Request Process in APP

Input:
c: Current node
$SeqNum$: Sequence Number of The received packet
$LastReceived[i]$: Sequence number of the last packet received for the substream i
$PackSourceNode$: Node ID of the sender of the received packet
$MaxMissing$: Maximum number of packets that can be recovered
t: Current node local time in milliseconds (ms)
Begin
$SubStream \leftarrow SeqNum \bmod NumberOfSubstream$
if $SeqNum - LastReceived[SubStream] > NumberOfSubstreams$ **then**

$MissingNumber = \min(MaxMissing, \frac{SeqNum - LastReceived[SubStream]}{NumberOfSubstreams} - 1)$

create a Request Packet $ReqPackFromPusher$
create a Request Packet Array $ReqPackFromParents$
for $i = 0$ **to** $MissingNumber - 1$ **do**
Insert packet $(SeqNum - (i+1) \times NumberOfSubstreams)$ into $ReqPackFromPusher$
schedule to send $ReqPackFromPusher$ at time $t + 1$;
for each parent $p \neq PackSourceNode$ **do**
Insert packet$(SeqNum - (i+1) \times NumberOfSubstreams)$ $ReqPackFromParent[p]$
end for
schedule to send $ReqPackFromParent$ at time $t + 201$; {the waiting time $201ms$ may be changed according to the RTTs of c with its parents to provide enough time to the packets that have been requested from the pusher to arrive}
end for
end if
End

Algorithm 9.9 Request Packet Processing on Reception in *BPP*

Input:
p: Current node
Begin
for each packet *pack* being requested **do**
  if $p.inBuffer(pack)$ **then**
    generate RND randomly (1 to 100)
    if $RND \leq 100 \times RPloss_{BPP}$ **then**
      send requested packet
    end if
  end if
end for
End

Algorithm 9.10 Request Packet Processing on Reception in *APP*

Input:
p: Current node
$PackSourceNode$: Node ID of the sender of the received request
t: Current node local time in milliseconds (ms)
Begin
for each packet *pack* being requested **do**
  if $p.inBuffer(pack)$ is true **then**
    if $t > p.received\_time(pack) + rtt(p, PackSourceNode)$ **then**
      if $p.isPusherFor(PackSourceNode, Substream)$ is true **then**
        send requested packet
      else
        generate RND randomly (1 to 100)
        if $RND \leq 100 \times RPloss_{APP}$ **then**
          send requested packet
        end if
      end if
    end if
  end if
end for
End

Due to the fact that these steps will be executed by each parent of c, in both *BPP* and *APP*, we use relay probabilities, RP_{BPP} and RP_{APP}. This way, we can reduce the redundancy of packets. Discussion about the value of RP_{BPP} and RP_{APP} is made in Section 9.3.3.

In *BPP* and *PushPull*, the push group ID is set to the same value for each node and for each substream, see Section 9.2. So, to make push decisions, a node needs to wait for the packets that belong to that group. Delayed push decisions increase the redundancy because all parents of a node will relay the received packets of a substream until c makes a push subscription.

In *APP*, we manage to speed up the start up process while reducing the startup redundancy by assigning for each receiving node c a different value of the group ID for each received substream. This way, push subscriptions are faster, which results in lower redundancy.

Child Side: When a peer c sends a request to a peer p to be its parent, it adds p in its list of parents and waits for receiving packets from p. If the received packet is a request refusal, then c removes p from its parent list. When c receives a streaming packet, it first determines the substream of the packet. Then, if the packet belongs to the group on which push decisions are based, there are two possible situations (inherited from the original push-pull algorithm of [ZZSY07]):

- c is missing a pusher for the related substream either because it is in a startup phase or because it canceled a subscription to the old pusher. In this case, c sends a subscription request to p and updates the pusher list.

- The pusher of the concerned substream is different from the packet sender. If the packet initiates a push cycle then the current substream pusher is replaced by the packet sender. A push cancel request is sent to the current pusher while

a push subscription request is sent to the packet sender.

In *BPP* and *APP*, we impose that push decisions must not be based on requested packets. In addition, each peer c makes a periodic monitoring of its pusher map. If one substream has no pusher, c determines its worst parent. The worst parent is the one with the lowest traffic volume to c. If its transfer volume to c is less than the size of one substream, then, c eliminates it and looks for a new one.

9.3.2 Loss Recovery

Detection and Request.

To detect packet loss, a peer c monitors the sequence number of the received streaming packets. S being the number of substreams, packets of the same substream have numbers of the form $k + i \times S$, where k, i are integers and $k < S$. If peer c receives both packets with IDs n and $n + i \times S$, then it concludes that the corresponding pusher is working fine but the packets with ID $n + j \times S, 0 < j < i$ have been lost. These IDs are added to the list of missing packets.

In *BPP*, the list of missing packets will be sent to all parents. It is an explicit request of missing packets. Such a request is sent periodically. A lost packet is requested one time only. Parents who receive such requests will send the missing packets if they have them in their buffers. Here again, a parent will send a packet with the probability RP_{BPP} to reduce redundant traffic, see Algorithm 9.7.

In *APP*, see Algorithm 9.8, we split the request process into two steps based on the fact that we are dealing with packet loss and not with parent failure. Indeed, when a node c detects that it did not receive a specific packet *pack*, only two situations are possible. Either *pack* was lost on the overlay link between its pusher p and the receiver c, or p itself did not receive *pack*. In the first situation, a simple request to p is needed and has a high success ratio as the probability that the same packet is

lost two successive times or on two successive overlay links is very low. In the second situation, the request to p will be unsatisfied. However, if p receives *pack*, it will relay it to c right away as it does for substreams for which c has subscribed with it.

In the case where the request to the pusher proved to be unsuccessful, APP uses the multi-parent ability of the mesh by requesting the lost packets from its parents, excluding the designated pusher.

Request Processing.

When a node p receives a request for a lost packet *pack* from a node c, it checks if it has it. If the packet is still in its buffer, then p is able to satisfy the request. In BPP, p will send the requested packet with a a probability $RPloss$. As in BPP, we do not differentiate among parents, we set $RPloss_{BPP} = RP_{BPP} = 1/NumberOfParents$, see Algorithm 9.9.

On the opposite, APP does differentiate among parents, see Algorithm 9.10. p determines the substream of *pack* and then checks if c has subscribed to this substream with it. If it is the case, the packet is sent to c. Otherwise, the packet is sent with a low probability $RPloss_{APP}$.

9.3.3 Tuning

BPP.

The relay probability RP_{BPP} is the most critical parameter for the performance of the probabilistic push strategy. A too high RP_{BPP} is likely to result in a high redundancy ratio while a too low value will result in a low success rate for requested packets and a long start up. The events of peers sending packets to a same receiver being independent, the probability that a peer c with n parents receives exactly one copy

of the same packet, $P_{c,1,n}$ can be written as follow:

$$P_{c,1,n} = \binom{1}{n} \times RP_{BPP}(1 - RP_{BPP})^{n-1}.$$

Let $D = 1 - RP_{BPP}$ be the probability that a node does not relay a packet. Then, we can write:

$$P_{c,1,n} = \binom{1}{n} \times (D^{n-1} - D^n).$$

Optimum values are obtained for:

$$(n-1) \times D^{n-2} - n \times D^{n-1} = 0.$$

And therefore $P_{c,1,n}$ is maximized when:

$$D = \frac{n-1}{n}.$$

Then the best value of the relay probability is equal to

$$RP_{BPP} = \frac{1}{n}.$$

APP.

BPP is a basic version of the probabilistic push strategy. More developed versions can be proposed. For instance, the relay probability may be different from one parent to another, depending on its performance. *APP* implements such an idea by differentiating among potential good pushers for a specific substream, at both startup and loss recovery.

At startup, the relay probability RP_{APP}, see Algorithm 9.6, is likely to be different from a parent to another. The maximum value of RP_{APP} is equal to RP_{BPP} as

computed in the *BPP* version. Then, the *APP* algorithm assigns a different value to favor the choice of potentially better pushers, for a given substream s and a receiving node. *APP* will prefer a potential pusher having:

- a low RTT to the concerned child c,
- high available upload slots,
- already a pusher for substream s.

A low RTT with c would favor the use of low delay links and would speed up the startup process and the delivery. A high number of available upload slots means that the potential pusher p would accept a pushing subscription from node c if needed. The fact that p has already a pusher for substream s means that, unless there is a problem with its pusher, peer p would be provided with the substream content on a regular basis.

Algorithm 9.6 shows how we exploit these criteria. Parents that are judged not to be good pushers for a given substream and a given child, will still relay the packet with a low probability:

$$RP_{BPP}/NumberOfParents.$$

9.3.4 Summary of advantages

The pure push strategy presents the following advantages:

- Low Overhead: There is no buffer map exchanges among nodes which is usually the main cause for the overhead along with redundancy.
- No use of the pull mechanism at the start up or at packet loss. It is replaced by a probabilistic push.
- Low Redundancy: It only occurs for lost packets or at the startup.

- Rapidity: Parents start sending before receiving push requests. As soon as a node becomes a child of another, it starts receiving packets of different sub-streams from it.

- Simplicity: There is no decision process on what to push. A packet is pushed as soon as it is received if there is a spare space in the upload buffer.

- Resiliency to inaccurate information: The inaccuracy of push map information has less impact than than a buffer map inaccuracy.

9.4 Simulations

The objective of the simulations is to measure the impact of the different scheduling strategies on the average playback delay observed by peers in the streaming session. In the following, we will describe the simulation environment and scenarios we are using. We use the upload resource organization schema presented in Section 6.2.

9.4.1 Simulator

We make use of the discrete event based simulator for P2P live streaming built by Zhang *et al.* and detailed in [ZZSY07]. This simulator operates at the packet level and its source code is available at [Zha09].

Some parameter values are detailed in Table 9.1. Network end-to-end latencies (RTT) are real-world values taken from a latency matrix computed within the Meridian project [WSS05].

In the simulator, the streaming data is divided into packets of size 1250 bytes, not including headers. For each algorithm, we execute the simulation three times and take the average values. Such number is sufficient: As we are not considering churn rate at the startup of the session, the overlay construction algorithm leads to

Table 9.1: Simulation Parameters

Streaming Rate	$200kbps$
α: Slot overhead/retransmissions	0.25
Needed Parents per Node	4
Missing Packets Request Period	$200ms$
Push Map Monitoring Period	$5,000ms$
Sending Buffer Map Period (push-pull only)	$1,000ms$

the same overlay for all the strategies being considered. We limit each execution to a duration of 180 seconds. This duration does not impact the final results as the system reaches its steady state in less than 100 seconds. The results shown are the ones computed for the last 10 seconds.

9.4.2 Scenario

The BP-FP Peering Strategy.

We next describe the peering strategy used in the simulations. The ***BP-FP*** strategy has been proposed in [OKJ09a]: A node looks for exactly P parents that will provide it with shortest paths to the source node. A path length is the sum of all link delays and depends on the end-to-end RTT among peers.

We have improved the overlay construction algorithm through the following:

- a child of the source receives the full stream from it.
- a child whose upload bandwidth is less than the streaming rate cannot be a child of the source.

Node Information.

We have conducted simulations for the six sets of nodes from Table 9.2. These nodes have typical real world values of the upload bandwidths. For each set, the slot index, (SI), is the fraction of all upload slots made available by peers to the total number

Table 9.2: Upload Bandwidth Capacities of Nodes

Clients (N)	1 Mbps (%)	384 kbps (%)	128 kbps (%)	S (kbps)
51	10	13	77	1,000
101	10	20	70	1,000
201	10	25	65	1,000
401	10	25	65	3,000
701	10	25	65	5,000
1,001	10	25	65	8,500

of needed slots for all nodes and is equal to 1.14. It is assumed that nodes have DSL connections and that bandwidth bottlenecks are located on the edge of the network, i.e., end-node access networks [ZZSY07]. Table 9.2 shows how bandwidth capacities are distributed.

Nodes join the session following an identical arrival rate for the three node types.

Packet Loss.

We have considered two situations. The first one is an ideal case where there are no lost packets. In the second situation, each peer in the session is experiencing a streaming packet loss rate of 2%. This is a typical value as measurements in [WHLR09] show.

9.5 Results

In this section, we present and comment the results obtained through the simulations. *PurePush-LF*, *BPP-LF* and *APP-LF* (respectively *PurePush-LR2*, *BPP-LR2* and *APP-LR2*), designates the compared strategies in the situation without packet loss (respectively, with packet loss). All strategy variations led to 100% in reception quality in the ideal case and higher than 98.9% in the loss case.

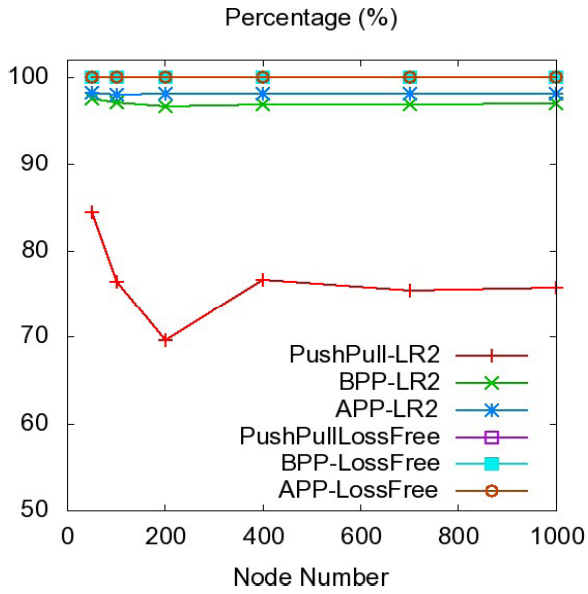


Figure 9.1: Average Push Rate, without Churn Rate

9.5.1 Average Push Rate

The average push rate is the average, over all nodes, of the rate of packets that have been pushed (vs. requested). The push rates of the different strategies are depicted in Figure 9.1. All the strategies led to a high push packet rate ($\geq 80\%$), for all scenarios. We note that *APP-LF*, *BPP-LF* and *PushPull-LF* led to the same push rate (100%). The reason is that the push-pull strategy has managed to construct a distribution tree for each substream and thus all packets are pushed. As a consequence, we expect that all loss free variations will lead to similar playback delay performances. *APP-LR2* is second best but performs very well as the pushing rate is always greater than 98%. BPP-LR2 leads to a slightly lower pushing rate. It is less than 100% for both, because we count neither redundant nor explicitely requested packets as pushed packets. *PushPull-LR2* performs the worst by far. Thus, the pull mechanism is very active and, as we will see in Section 9.5.2, the playback delay of such a strategy will be high comparatively to the others.

9.5.2 Average Playback Delay

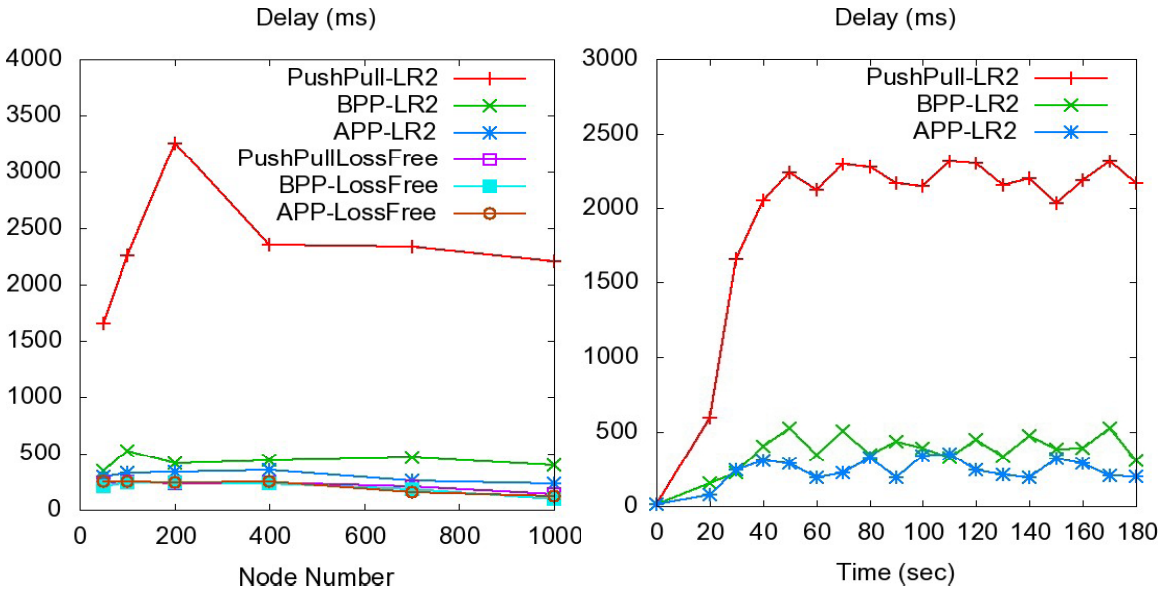
We define the average playback delay as the average of playback delays of all nodes except the source. As in [ZZSY07], the playback delay depends on the ratio r of the received packets to the needed packets per buffer intervals of one second. Every two consecutive intervals have a time slot lag of $250ms$. The playback delay is then determined by the lowest i^{th} time slot for which the corresponding buffer interval satisfies $r \geq 99\%$. The playback delay is then equal to $i \times 250ms$, i starting at 160 downward 0.

The average playback delays of the peers, at the instant 180 seconds, are depicted in Figure 9.2a. We note that the *BPP-LF*, *APP-LF* and *PushPull-LF* perform the best and lead to very similar performance as there is no packet loss.

APP-LR2 performs second best and leads to low playback delays (between 250ms and 120ms for all scenarios). This means that it succeeds at recovering lost content quickly and efficiently. *BPP-LR2* does the same with a slightly lower success than *APP-LR2*.

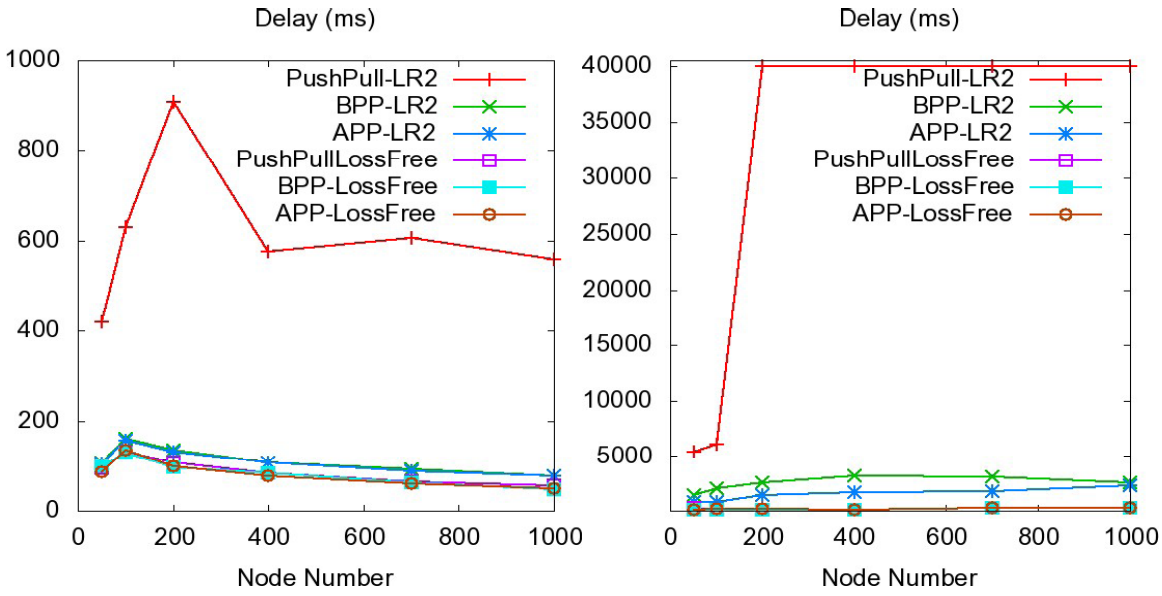
The fact that *PushPull-LR2* has the worst performance by far confirms that push-pull strategies are not suitable for applications needing low delays in the real context because packet loss is a common event.

As we are dealing with random packet loss, we expect that the playback delay performance will vary from time to time. Thus, for each strategy, we consider the best execution with $N = 1001$ (the one leading to the lowest average playback delay). Then, in Figure 9.2b, we depict the playback delays, obtained at each 10 seconds time period starting at the instant 20 seconds to 180 seconds. Results confirm the observations we already made at the beginning of the current section.



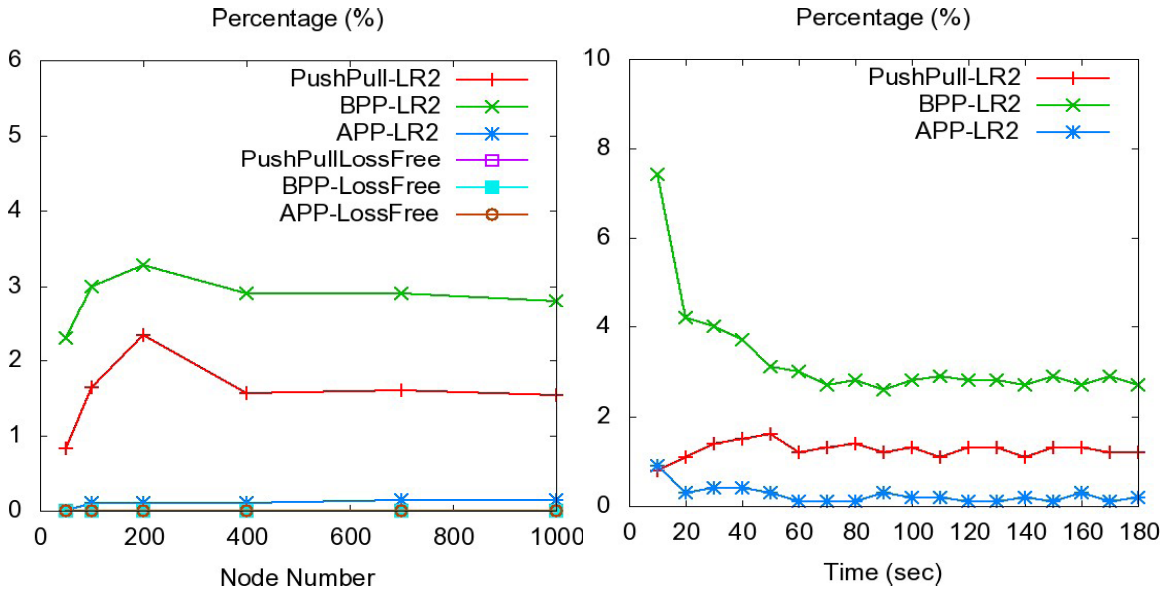
(a) Average Playback Delay at t=180s (b) Average Playback Delay, Session $N = 1001$

Figure 9.2: Playback Delays, without Churn Rate



(a) Average Packet Delay (b) Average Maximum Playback Delay

Figure 9.3: Other Delay Results, without Churn Rate



(a) Average Redundancy at t=180s (b) Redundancy,Session $N = 1001$

Figure 9.4: Average Redundancy, without Churn Rate

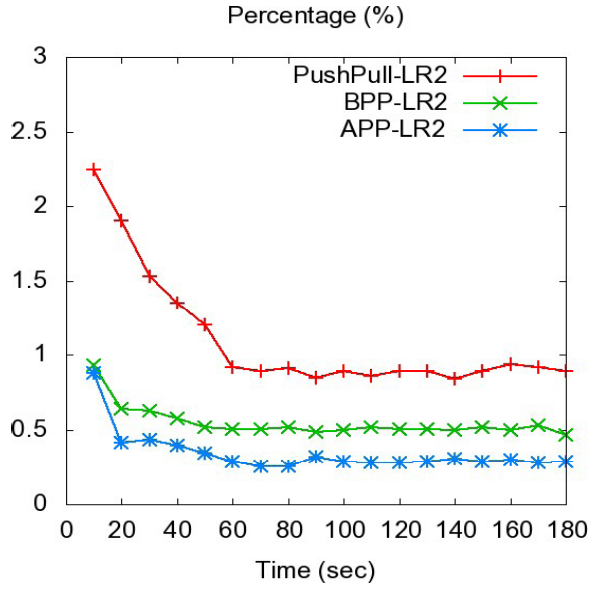


Figure 9.5: Average Overhead, Session $N = 1001$, without Churn Rate

9.5.3 Average Packet Delay

The global average packet delay is obtained by computing the mean of average packet delay at all nodes except the source. It is calculated the same way as in [ZZSY07]. The packet delay is made of the scheduling delay and of the delivery delay.

As shown in Figure 9.3a, *APP-LR2* and *BPP-LR2* have the same packet delay. It is slightly higher than the LF variations due to the longer delay needed to recover lost packets. It is clear that the pure push approach recovers lost packets quickly: A node does not wait for buffer maps to request a packet. In addition, in the case of interactive application, the average packet delay may be regarded as a more relevant metric than the playback delay as defined in Section 9.5.2. The obtained values are very low, less than $100ms$ in most cases, and confirm the fact that push scheduling is suitable for some interactive applications.

9.5.4 Maximum Playback Delay

The maximum playback delay results are presented in Figure 9.3b. *APP-LR2* has a higher maximum than the *LF* variations. This is due to the fact that lost packets up in the mesh (near the source) either will take longer time or will never reach to nodes down in the distribution trees. The same problem causes *PushPull-LR2* to perform the worst but it is amplified by the pull mechanism.

9.5.5 Redundant traffic

The redundant traffic is defined as the ratio of the number of redundant packets received by a node to the total received streaming packets. The average redundancy is the average of the redundant traffic over all nodes. In Figure 9.4a, we depict the redundancy rate obtained at time $t = 180s$. At that time, the system has already reached a steady state and thus redundancy is mainly due to the request of lost

packets.

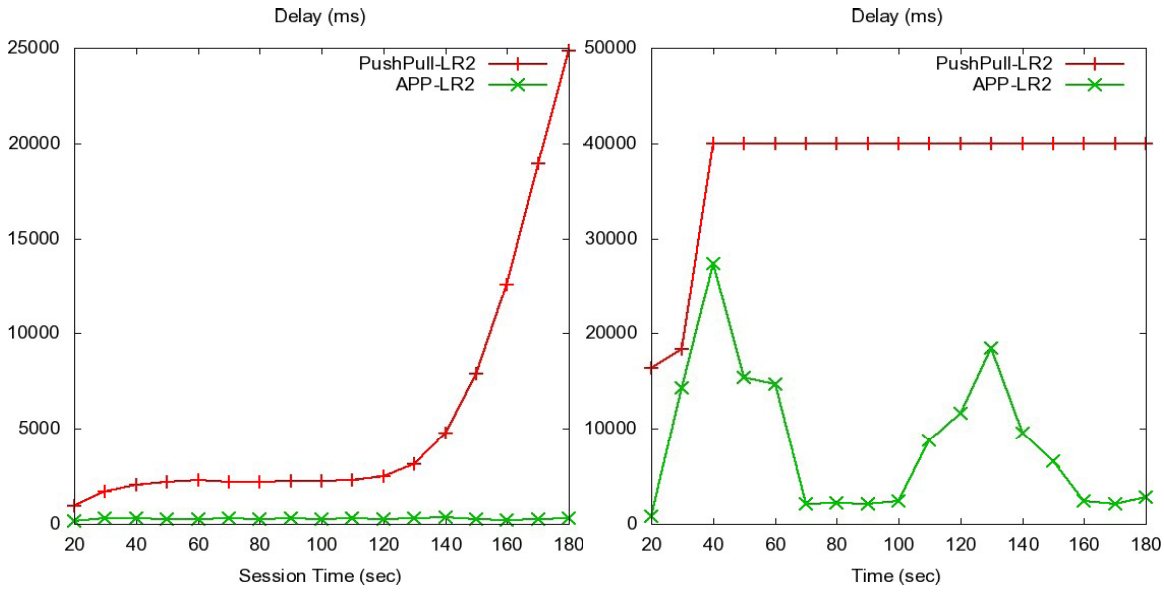
APP-LR2 has a very low redundancy thanks to the two steps of the request process of missing packets: Firstly, to request from the assigned pusher and then, if content is still missing, to request from the other parents. *BPP-LR2* leads to the highest redundancy because it requests a lost packet from all parents at the same time. *PushPull-LR2* is more efficient in terms of redundancy because the requests are based on content availability at parents.

Figure 9.4b shows how redundant traffic varies during a session. *BPP-LR2* experiences a high redundancy at startup that decreases until the session reaches a steady state. *APP-LR2* has the lowest redundancy. It is effective at speeding up the startup process and thus at reducing the use of the probabilistic push. This is helped by the fact that it assigns different relay probabilities to parents depending on their potential to be pushers of a specific substream to a given child. *PushPull-LR2* has higher redundancy because the delay for retrieving a lost packet is sometimes higher than the request period. Also, re-scheduling substreams may lead to redundancy as noted in [LYHT08].

9.5.6 Overhead traffic (Session)

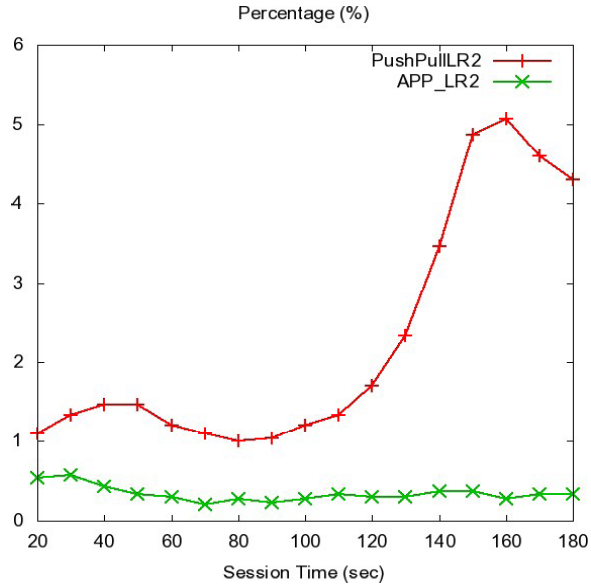
In Figure 9.5, we depict the overhead experienced by the system during a session. *BPP-LR2* and *APP-LR2* lead to lower overhead than *PushPull-LR2* because we eliminate the mandatory exchange of buffer maps and we replace it by a push map sending when necessary. Information is thus exchanged at smaller size and frequency. In addition, *BPP-LR2* and *APP-LR2* request a missing packet once at most from each parent.

All strategies experience some overhead increase at startup because of the information exchange. *BPP-LR2* has higher overhead than *APP-LR2* because of its



(a) Average Playback Delay

(b) Average Maximum Playback Delay



(c) Average Redundancy

Figure 9.6: Results for Sessions with 1001 Nodes and 10% Churn Rate

request process in which it sends the same request to all its parents at the same time.

9.5.7 Impact Of Churn Rate

To measure the impact of the peer churn rate on the performance, we impose that 10% of nodes leave the session at a random time between the instants $100s$ and $160s$. In such a context, we only consider the *APP* and the *Push-Pull* strategies. For each strategy, we repeat the execution three times and then take the average values. The results are depicted in Figure 9.6. For all the strategies and the executions, the nodes leaving the session are the same but the departure times are random. The departues are abrupt, i.e., nodes do not inform their neighbors when leaving.

Figure 9.6a shows that the *push-pull* strategy is the one that suffers the most from the churn rate. The average playback delay increases exponentially. *APP* is more resilient to churn rate as the average playback does not change significantly.

To better show the impact of the churn, we refer to Figure 9.6b. For *APP*, we have two peaks. The first one is due to the start up process of nodes. The second one happens after the steady state has been reached and is due to the churn rate. However, the increase is less significant than with the *Push-Pull* strategy and more importantly. we observe that the system is recovering from the node departures as the maximum playback delay is decreasing with time.

It is also worth noting that with the *Push-Pull* strategy, the number of affected nodes is big and continuously growing. Indeed, the maximum playback delay cannot go beyond $40s$ (limit imposed by the simulator design and equivalent to undefined delay) and the average playback delay increases continuously. The fact that the average playback delay of the *APP* strategy does not show a significant peak (although the maximum playback delay does) means that the number of affected nodes remains small.

With respect to redundancy, *APP* does not show any significant increase in redundancy. However, *Push-Pull* presents a significant increase in redundancy compared to the steady state and even to the startup state. This confirms the conclusion that the number of affected nodes is very big. Thus the pull mechanism is being used heavily which results in the redundancy increase.

9.6 Conclusion

In the present work, we point out that although push-pull scheduling leads to low playback delays in ideal situations, its performance degrades when dealing with packet loss or at the startup process. As P2P video applications tend to move toward interactivity, there is a need for a lighter but more efficient scheduling. We addressed this need by proposing the *PurePush* approach: Starting from a push-pull approach, we replaced the pull mechanism by a probabilistic push.

The most challenging part was to eliminate packet redundancy. We managed to achieve that in two stages: Speeding up the startup process and implementing a two-step lost packet request. The *Advanced Pure Push (APP)* variation led to low delays and even to lower redundancy and overhead than push-pull scheduling.

Although *APP* showed good resiliency for a parent loss rate equal to 10% some individual peers may experience high performance degradation. Indeed, there is no guarantee on either how long or how strong is the degradation. Thus, we plan to extend the *APP* variation with a mechanism that bounds both the degradation time and intensity.

Chapter 10

Conclusion and Future Work

The work described in this thesis aims at providing hints and solutions in order to minimize playback delays experienced by peers in a P2P live streaming session using a single source server. We have addressed two main aspects: the overlay construction and the content scheduling.

With respect to overlay construction, and beside proposing a strategy that performed the best in the scenarios we have considered, we had two main contributions. Firstly, we have showed the importance of relying on some information about the overlay network when selecting neighbors/parents. Such information relate to the delivery delays experienced by a node (path to the source) and the usage status of its upload bandwidth. We have called strategies that use such information, partially overlay-aware strategies (POS) in opposition to basic strategies (BS) which only rely on local information related to the characteristics of peers, such as RTTs and nominal upload bandwidth.

Secondly, we have proposed a MILP model to validate the results of overlay construction strategies. Due to the differences of scenarios either in simulation environments or a real-world context, it is difficult to evaluate a P2P system performance

independently of the design and implementation choices. None can claim that a specific strategy will always lead to the best performance.

However, a lot of work is still needed to stabilize the performance of a P2P streaming system. For instance such a system is sensitive to the join distribution of peers. Indeed, P2P systems have no reproducible behavior as they depend on some random events and they rely on many random decisions at different stages. Such an issue needs to be resolved if we want to deploy a P2P system that can be adopted by regular consumers. Here, the issues of quality of service, or the quality of experience, arise. Existing solutions try to alleviate these problems through increasing the available resources, or modifying the overlay. These are reactive solutions and we think that we need to propose proactive approaches in order to have a good control on the service degradation if it happens.

With respect to content scheduling, the fact that existing scheduling strategies in mesh overlays are based on knowledge of buffer maps of neighbors/parents make them less resilient to churn rate and packet loss. We addressed this issue by proposing a pure push scheduling algorithm where buffer map exchanges are completely eliminated. Although results show that *PurePush* has a good resiliency to churn and to packet loss, it does not offer any performance guarantee.

Such situation is annoying as, when deployed in a free P2P system or a commercial one, the consumer will not be satisfied and is likely not to use the system anymore. Indeed, it is very critical to propose solutions for quality guarantee if we want the use of P2P streaming systems to move further than streaming rights protected content for free.

In addition, the fact that *PurePush* leads to very low delivery delays opens wide possibilities for the use of P2P streaming systems with some non critical interactive applications. Thus, we are planning to implement *PurePush* in a real P2P system

and evaluate it with some interactive multimedia applications.

We also think that existing P2P streaming systems have focused mainly on the technical aspect. However, we believe that the social aspect is important as the P2P system has already the infrastructure to deploy strong social networking applications that can be combined with the video streaming functionality. For instance, each peer may act as a source server to stream a movie or to watch a live event with friends from the social network providing real time stream messages to share their thoughts or emotions.

However, here again rises the issue of content copyright. How to prevent a user from streaming a copyright protected content? How to ensure that the peers in the session have the rights to watch the video stream? These are still sensible issues that prevent content providers and consumers alike from adopting the P2P content distribution paradigm.

So far, P2P systems have never been considered as business opportunities that can generate profit for the ISPs. However, we think that with the G-PON (specially FTTH) networks being widely deployed and the IPTV/Triple Play (Phone + Internet + TV) services up and running, there is a room to use a P2P network of subscribers within the same provider or regional providers. The benefit lies in alleviating the load on the provider servers and the possibility to add the social networking service.

Bibliography

[ABE+09] J. Arkko, B. Briscoe, L. Eggert, A. Feldmann, and M. Handley. Dagstuhl perspectives workshop on end-to-end protocols for the future internet. *SIGCOMM Comput. Commun. Rev.*, 39(2):42–47, 2009.

[AFS07] V. Aggarwal, A. Feldmann, and C. Scheideler. Can isps and p2p users cooperate for improved performance? *SIGCOMM Comput. Commun. Rev.*, 37:29–40, July 2007.

[AKR+05] M. Adler, R. Kumar, K. Ross, D. Rubenstein, T. Suel, and D.D. Yao. Optimal peer selection for P2P downloading and streaming. In *INFOCOM*, pages 1538–1549, 2005.

[ATS04] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACOM Computing Surveys*, pages 335–371, 2004.

[BB04] S. Birrer and F. E. Bustamante. Resilient peer-to-peer multicast without the cost. Technical Report NWU-CS-04-36, CS Department, Northwestern University, January 22 2004.

[BCC+06] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang. Improving traffic locality in bittorrent via biased neighbor

selection. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 66, Washington, DC, USA, 2006. IEEE Computer Society.

[BCMR04] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. *Networking, IEEE/ACM Transactions on*, 12(5):767–780, 2004.

[BIT11] BITTORENT. `http://www.bittorrent.com`. 2011.

[BLBS03] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 102–113, New York, NY, USA, 2003. ACM Press.

[BLMR98] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 56–67, New York, NY, USA, 1998. ACM Press.

[BLPL+08] L. Bracciale, F. Lo Piccolo, D. Luzzi, S. Salsano, G. Bianchi, and N. Blefari-Melazzi. A push-based scheduling algorithm for large scale p2p live streaming. In *Telecommunication Networking Workshop on QoS in Multiservice IP Networks, 2008. IT-NEWS 2008. 4th International*, pages 1 –7, 13-15 2008.

[BMM+08] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg. Epidemic live streaming: optimal performance trade-offs. In *Proceedings of*

the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, SIGMETRICS '08, pages 325–336, New York, NY, USA, 2008. ACM.

[CB08] D. R. Choffnes and F. Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 363–374, New York, NY, USA, 2008. ACM.

[CCB07] D. Carra, R. L. Cigno, and E. W. Biersack. Graph based analysis of mesh overlay streaming systems. *Selected Areas in Communications, IEEE Journal on*, 2007.

[CDK+03] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 298–313, New York, NY, USA, 2003. ACM.

[CDKR02] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[CPL10] CPLEX. `http://www.ilog.com/products/cplex`, 2010.

[CRZ00] Yang-hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast (keynote address). In *SIGMETRICS '00: Proceedings*

of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 1–12, New York, NY, USA, 2000. ACM.

[CXH08] Z. Chen, K. Xue, and P. Hong. A study on reducing chunk scheduling delay for mesh-based p2p live streaming. In *Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing*, pages 356–361, Washington, DC, USA, 2008. IEEE Computer Society.

[Dee95] S. E. Deering. Multicast routing in internetworks and extended lans. *SIGCOMM Comput. Commun. Rev.*, 25:88–101, January 1995.

[EGmKM04] P. T. Eugster, R. Guerraoui, A. m. Kermarrec, and L. Massouli. From epidemics to distributed computing. *IEEE Computer*, 37:60–67, 2004.

[FCAB00] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *Networking, IEEE/ACM Transactions on*, 8(3):281–293, 2000.

[GLL08] Y. Guo, C. Liang, and Y. Liu. AQCS: Adaptive queue-based chunk scheduling for p2p live streaming. In Amitabha Das, Hung Keng Pung, Francis Bu-Sung Lee, and Lawrence Wai-Choong Wong, editors, *Networking*, volume 4982 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 2008.

[Goy01] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, 18(5):74–93, September 2001.

[GWYS05] X. Gu, Z. Wen, P. S. Yu, and Z. Y. Shae. Supporting multi-party voice-over-ip services with peer-to-peer stream processing. In *MULTIMEDIA*

'05: Proceedings of the 13th annual ACM international conference on Multimedia, pages 303–306, New York, NY, USA, 2005. ACM Press.

[HFPW03] M. Handley, S. Floyd, J. Pahdye, and J. Widmer. Tcp friendly rate control (tfrc): Protocol specification, rfc3448, proposed standard, 2003.

[HHB+03] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. Promise: peer-to-peer media streaming using collectcast. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 45–54, New York, NY, USA, 2003. ACM Press.

[hi10] http://www.future internet.eu, 2010.

[HRV09] F. Huang, B. Ravindran, and A. Vullikanti. An approximation algorithm for minimum-delay peer-to-peer streaming. In *Peer-to-Peer Computing, P2P '09. IEEE Ninth International Conference on*, pages 71–80, 2009.

[HSLG99] U. Horn, K. Stuhlmuller, M. Link, and B. Girod. Robust internet video transmission based on scalable coding and unequal error protection. *Image Com.*, 15(1–2):77–94, Sep 1999.

[KR07] R. Kumar and K. Yong Liu Ross. Stochastic fluid theory for p2p streaming systems. *Selected Areas in Communications, IEEE Journal on*, 2007.

[KRAV03] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of the 9th ACM SOSP Symposium*, pages 282–297, 2003.

[LGL08] C. Liang, Y. Guo, and Y. Liu. Is random scheduling sufficient in p2p video streaming? In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS 2008)*, 2008.

[Liu07] Y. Liu. On the minimum delay peer-to-peer video streaming: how realtime can it be? In *Proceedings of the 15th international conference on Multimedia*, MULTIMEDIA '07, pages 127–136, New York, NY, USA, 2007. ACM.

[LJL+06] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng. Anysee: Peer-to-peer live streaming. In *Proc. of INFOCOM*, April 2006.

[LLC09] D. Liu, F. Li, and S. Chen. Towards optimal resource utilization in heterogeneous p2p streaming. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 352–359, Washington, DC, USA, 2009. IEEE Computer Society.

[LLR09] C. Liang, Y. Liu, and K. W. Ross. Topology optimization in multi-tree based p2p streaming system. In *Proceedings of the 2009 21st IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '09, pages 806–813, Washington, DC, USA, 2009. IEEE Computer Society.

[LMSS01] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.

[LMSW07] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer. Push-to-Pull Peer-to-Peer Live Streaming. In *21st International Symposium on Distributed*

Computing (DISC), Lemesos, Cyprus, Springer LNCS 4731, September 2007.

[LN06] J. Liang and K. Nahrstedt. Dagstream: locality aware and failure resilient peer-to-peer streaming. In *13th SPIE/ACM MMCN Conference*, volume 6071, January 2006.

[LYHT08] Z. Li, Y. Yu, X. Hei, and D. H. K. Tsang. Towards low-redundancy push-pull p2p live streaming. In *QShine '08: Proceedings of the 5th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 1–7, 2008.

[LYKM06] Z. Liu, H. Yu, D. Kundur, and M. Merabti. On peer-to-peer multimedia content access and distribution. In *Proc. IEEE International Conference on Multimedia and Expo, Toronto, Canada*, 2006.

[MAM06] M. Mushtaq, T. Ahmed, and D. E. Meddour. Adaptive packet video streaming over p2p networks. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 59, New York, NY, USA, 2006. ACM Press.

[MR06] N. Magharei and R. Rejaie. Understanding mesh-based peer-to-peer streaming. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, May 2006.

[MR09] Nazanin Magharei and Reza Rejaie. Prime: peer-to-peer receiver-driven mesh-based streaming. *IEEE/ACM Trans. Netw.*, 17:1052–1065, August 2009.

[NAP06] NAPSTER. `http://www.napster.com`. 2006.

[OKJ09a] A. Ouali, B. Kerherve, and B. Jaumard. Revisiting peering strategies in push-pull based p2p streaming systems. In *Multimedia, IEEE International Symposium on*, pages 350–357, 2009.

[OKJ09b] Anis Ouali, Brigitte Kerherve, and Brigitte Jaumard. Toward new peering strategies for push-pull based p2p streaming systems. In *Ultra Modern Telecommunications & Workshops, 2009. ICUMT '09. International Conference on , P2PNet'09*, 2009.

[OKJ11] Anis Ouali, Brigitte Kerherve, and Brigitte Jaumard. A packet-loss resilient push scheduling for mesh overlays. In *IEEE Consumer Communications and Networking Conference*, 2011.

[Ooi04] W. T. Ooi. Dagster: contributor-aware end-host multicast for media streaming in heterogeneous environment. In S. Chandra and N. Venkatasubramanian, editors, *Multimedia Computing and Networking 2005. Edited by Chandra, Surendar; Venkatasubramanian, Nalini. Proceedings of the SPIE, Volume 5680, pp. 77-90 (2004).*, pages 77–90, December 2004.

[PKT+05] V. S. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *IPTPS*, pages 127–140, 2005.

[PM08] F. Picconi and L. Massoulie. Is there a future for mesh-based live video streaming? In *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, pages 289–298, 2008.

[PPL10] PPLIVE. `http://www.pplive.com/en`, 2010.

[PPS06] PPSTREAM. `http://www.ppstream.com`. 2006.

[PWCS02] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 177–186, New York, NY, USA, 2002. ACM Press.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[Rip01] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the 1st Int'l Conf. on Peer-to-Peer Computing*, pages 99–100, 2001.

[RLC08] D. Ren, Y.-T. H. Li, and S.-H. G. Chan. On reducing mesh delay for peer to peer live streaming. In *Proc. of IEEE INFOCOM*, 2008.

[RLC09] Dongni Ren, Yui-Tung Hillman Li, and S.-H. Gary Chan. Fast-mesh: a low-delay high-bandwidth mesh for peer-to-peer live streaming. *Trans. Multi.*, 11:1446–1456, December 2009.

[RM05] A. El Rhalibi and M. Merabti. Agents-based modeling for a peer-to-peer mmog architecture. *Comput. Entertain.*, 3(2):3–3, 2005.

[RO03] R. Rejaie and A. Ortega. Pals: peer-to-peer adaptive layered streaming. In *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 153–161, New York, NY, USA, 2003. ACM Press.

[RS04] R. Rejaie and S. Stafford. A framework for architecting peer-to-peer receiver-driven overlays. In *NOSSDAV '04*, pages 42–47, 2004.

[SGMZ04] K. Sripanidkulchai, A. Ganjam, B. Maggs, and Hui Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *SIGCOMM '04*, pages 107–120, 2004.

[SLL06] T. Small, B. Liang, and B. Li. Scaling laws and tradeoffs in peer-to-peer live multimedia streaming. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 539–548, New York, NY, USA, 2006. ACM Press.

[SNG05] E. Setton, J. Noh, and B. Girod. Rate-distortion optimized video peer-to-peer multicast streaming. In *P2PMMS'05: Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming*, pages 39–48, New York, NY, USA, 2005. ACM Press.

[Sop10] Sopcast. http://www.sopcast.org, 2010.

[SR06] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 189–202, New York, NY, USA, 2006. ACM.

[THD03] D. A. Tran, K. A. Hua, and T. T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE INFOCOM*, 2003.

[VYF06] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *Proceedings*

of the Proceedings of the 2006 IEEE International Conference on Network Protocols, pages 2–11, Washington, DC, USA, 2006. IEEE Computer Society.

[WHLR09] A. Wang, C. Huang, J. Li, and K. W. Ross. Queen: Estimating packet loss rate between arbitrary internet hosts. In *PAM*, pages 57–66, 2009.

[WL05a] C. Wu and B. Li. Optimal peer selection for minimum-delay peer-to-peer streaming with rateless codes. In *Proceedings of the ACM P2PMMS Workshop*, pages 69–78, 2005.

[WL05b] C. Wu and B. Li. rStream: Resilient peer-to-peer streaming with rateless codes. In *Proceedings of the 13th annual ACM Int'l conf. on Multimedia*, pages 307–310, 2005.

[WL07] M. Wang and B. Li. R2: Random push with random network coding in live peer-to-peer streaming. In *in IEEE Journal on Selected Areas in Communications, Special Issue on Advances in Peer-to-Peer Streaming Systems*, pages 1655–1666, 2007.

[WSS05] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: a lightweight network location service without virtual coordinates. *SIGCOMM Comput. Commun. Rev.*, 35(4):85–96, 2005.

[WXL10] F. Wang, Y. Xiong, and J. Liu. mTreebone: A collaborative tree-mesh overlay network for multicast video streaming. *Parallel and Distributed Systems, IEEE Transactions on*, 21(3):379 –392, 2010.

[XYK+08] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz. P4P: provider portal for applications. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 351–362, New York, NY, USA, 2008. ACM.

[YCM+04] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R.Y. Wang. Overlay mesh construction using interleaved spanning trees. In *INFOCOM*, volume 1, March 2004.

[YP05] C.-C. Yeh and L. S. Pui. On the frame forwarding in peer-to-peer multimedia streaming. In *P2PMMS'05: Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming*, pages 1–10, New York, NY, USA, 2005. ACM Press.

[ZCLB09] G. Zheng, S.-H. G. Chan, X. Luo, and A. C. Begen. Pattern-push: A low-delay mesh-push scheduling for live peer-to-peer streaming. In *ICME*, pages 1158–1161, 2009.

[Zha09] Meng Zhang. `http://media.cs.tsinghua.edu.cn/~zhangm`. June 2009.

[ZLLY05] X. Zhang, J. Liu, B. Li, and Y.S.P. Yum. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of IEEE INFOCOM*, pages 2102–2111, 2005.

[ZLZY05] 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 *Proceedings of the 13th annual ACM int'l conf. on Multimedia*, pages 287–290, 2005.

[ZSXY08] M. Zhang, L. Sun, X. Xi, and S. Yang. iGridmedia: Providing delay-guaranteed peer-to-peer live streaming service on internet. In *GLOBECOM*, pages 1741–1745, 2008.

[ZZSY07] M. Zhang, Q. Zhang, L. Sun, and S. Yang. Understanding the power of pull-based streaming protocol: Can we do better? *IEEE Journal on Selected Areas in Communications*, 25(9):1678–1694, 2007.

[ZZT+05] M. Zhang, L. Zhao, Y. Tang, J.-G. Luo, and S.-Q. Yang. Large-scale live media streaming over peer-to-peer networks through global internet. In *P2PMMS'05: Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming*, pages 21–28, New York, NY, USA, 2005. ACM Press.