PhD Dissertation

International Doctorate School in Information and Communication Technologies

Department of
Information Engineering and Computer Science
University of Trento, Italy

Cooperative Push/Pull Protocols for Live Peer-Assisted Streaming

Alessandro Russo

Advisor:

Prof. Renato Lo Cigno

University of Trento, Italy

April 2013

A mio padre

Abstract

Video streaming is rapidly becoming one of the key services of the Internet. Most streaming is today "on demand" and delivered via unicast delivery; however, many applications require delivery to many end-users and the lack of ubiquitous IP multicast remains a weakness of the Internet.

Given this scenario, the peer-to-peer (P2P) or peer-assisted communications is an appealing solution, especially in light of its intrinsic scalability and its extremely low initial investment requirements. However, the design of efficient, robust, and performing P2P streaming systems remains a high challenge, in particular when real-time (hard or soft) constraints are part of the service quality, as in TV distribution or conferencing.

This thesis deals with P2P live streaming, concentrating on unstructured, swarm-based systems. The protocols explored and proposed are based in general on mixed Push/Pull phases, i.e., the behavior of peers alternates between offering content to other peers and seeking the content from other peers.

The first part of the work is dedicated to the analysis of the fundamental properties of the Push/Pull protocols, including the enhancement of base protocols with a chunks' negotiation phase, which enable peers to execute parallel communications at the same time, fully exploiting their resources and drastically reducing duplicates and waste. Next, the focus is shifted on the impact of network parameters in video streaming distribution, showing that promoting locality in interactions leads to better performance than selecting target peers randomly.

Then, the attention is focused on wireless scenarios by mixing local multicast techniques (based on a modified version of the Protocol Independent Multicast –PIM– adapted to wireless environments) with active Pull recovery of missing data, with a peer-assisted approach. This protocol, called PullCast*, enables end-users to pull missed data packets via unicast communications while they receive video packet in multicast via push, exhibiting interesting results in terms of chunks diffusion delay and fraction of end-users served.*

Finally, the GRAPES library is introduced to provide a set of open-source components conceived as basic building blocks for developing new P2P streaming applications which have in mind the intelligent usage of network resources as well as the Quality of Experience of final users. GRAPES is the core library behind PeerStreamer, an open source P2P media streaming framework developed under the NAPA-WINE European research project, and currently supported by the EIT ICT Labs.

Keywords: Peer-to-Peer, Content Distribution, Multicast, Live Streaming, Push/Pull Protocols, Wireless Mesh Networks

Acknowledgments

The works presented in this thesis was supported by the European Commission through the NAPA-WINE Project (www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, Grant Agreement no.: 214412.

This doctoral thesis would not have been possible without the help and support of the kind people around me, and this page is here to thank them all.

First, I would like to express my sincere gratitude to my advisor, Prof. Renato Lo Cigno for being an excellent advisor and an outstanding person. His vision, knowledge, and advice have been a source of inspiration for me during my doctoral program, but also for the Master's degree. I would also like to thank him for the countless coffees and interesting discussions.

I would like to express my deep appreciation to Prof. Izhak Rubin who allowed me to visit him at the University of California, Los Angeles. You have been an excellent guide who enriched my knowledge with your exceptional insights into engineering.

I am deeply indebted to my committee members Prof. Marco Mellia and Prof. Pascal Felber for their time and effort in reviewing this work, providing invaluable suggestions.

A special thanks also goes to Csaba, Danilo, Roberta, Gianluca, and all the members of my group for their helpful support in these years.

I would like to thank all the friends I met at UCLA, Sina, Fabio, Philip, Alejandro, and especially Giuseppe and Yuxuan for all the racquetball games we enjoyed.

Vorrei ringraziare tutti i compagni di avventura con cui ho condiviso gioie e dolori del dottorato. Grazie a Christian, Nicola S., Andrea, Nicola M., Igor, Luca, Thomas, Edoardo, Chiara, Mattia, Nicola P., Valentina, Stefano S., Paolo, Stefano B., Gabriela, Yuri e Sasha.

Desidero ringraziare inoltre tutti gli amici siculi. Compagni di granite la domenica mattina. Anche se mi sono allontanto da casa la nostra amicizia non ne ha risentito. Mi siete stati sempre vicini e ritengo di essere molto fortunato ad avervi come amici. Grazie Pippo, Alfio, Salvo, Ivan e Giuseppe.

Un ringraziamento speciale va alla mia famiglia che da sempre mi accompagna e mi sostiene con tanto affetto e calore. Grazie a Rita, Alfio, Barbara, Vera, Orazio ed Antonio. Un grazie anche ai nipotini Daenerys e Sebastiano che allieteranno l'estate 2013.

Indubbiamente un ruolo fondamentale in questi anni è stato svolto da mia moglie Angela. Grazie per avermi sempre ascoltato ed incoraggiato ad andare avanti. Grazie per aver appoggiato tutte le mie scelte, anche quando sapevi che avremmo sofferto (sei mesi), sei stata fondamentale per me. Scusami per quelle volte che ti ho trascurato a causa del lavoro (sai con chi prendertela). Grazie Amore Mio.

Questo mio lavoro è dedicato ad una persona unica e speciale, che tengo a ringraziare per tutto quello che ha fatto, che fa e che farà per i suoi figli. Grazie per avermi incoraggiato e sostenuto in tutta la mia vita. Grazie per avermi reso indipendente e responsabile nelle decisioni, lasciandomi sempre piena libertà di scelta, sostenendo non solo economicamente ma anche con infinito amore la mia scelta di proseguire gli studi lontano da casa. Grazie per avermi sostenuto nei momenti di tristezza e sconforto, in particolare i primi periodi a Trento. Grazie per la fiducia riposta in me. Sono diventato la persona che sono oggi soprattutto grazie a te. Grazie papà.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Content Distribution

In the last decade the Internet has experienced a tremendous growth of data, especially multimedia contents. According to Cisco VNI (Visual Networking Index), video streaming is dominating mobile communications, exceeding 51% of total traffic by the end of 2012,[1] accounting for over two-thirds of total mobile data traffic by the end of 2017, as reported in Fig. 1.1. Thus, video and TV services are fast becoming an important part of users' lives, resulting in a significant grown of bandwidth used for delivering video contents.

Broadband penetration is increasing dramatically; Telecommunication companies are deploying miles of optical fibers to improve the last-mile telecommunications from fiber-to-the-neighborhood (FTTN) to fiber-to-the-home (FTTH), offering more and more bandwidth to home and business networks; On the wireless side, the fourth generation of mobile communication (4G) offers high speed broadband Internet access, guaranteeing high mobility communications through the Long-Term Evolution (LTE), and the future LTE-Advanced.

[1] http://www.cisco.com

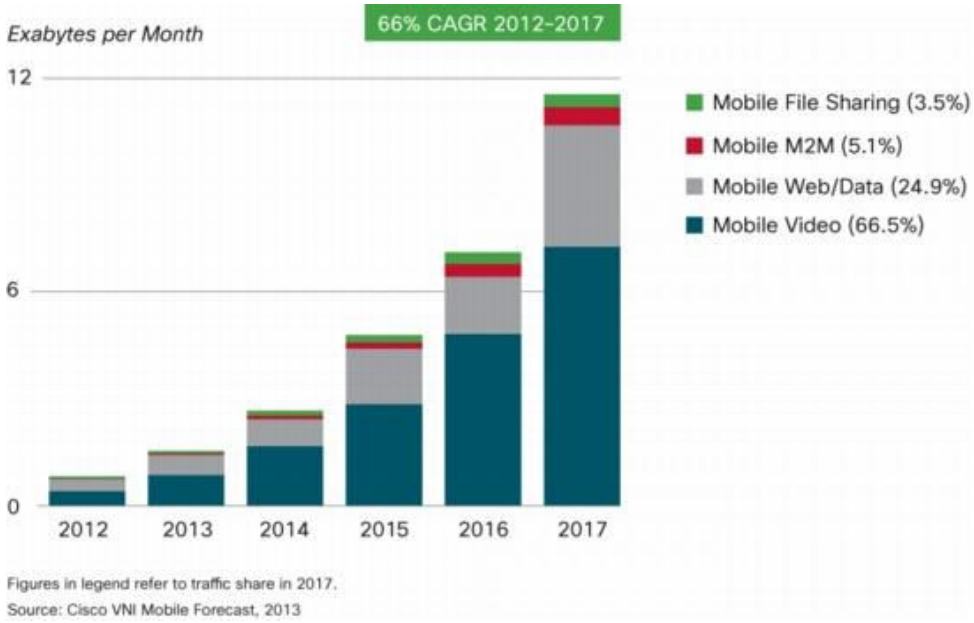

Figure 1.1: Mobile traffic forecast (Cisco VNI 2013).

TV broadcasters, independent producers, and the end-users themselves are willing to offer their video contents to the Internet community which represents a potential market that makes video streaming more attracting than ever.

Video streaming is a typical multicast application which involves a large number of participants that are willing to receive the same stream of packets. Content providers can rely on the client-server architecture to stream their video contents. Although this architecture is simple to manage and to update, however, it is very expensive to maintain and is not suitable for serving large population of users, which is a fundamental requirement in video streaming systems.

Today, the content distribution is a potential business opportunity; New companies are deploying their private infrastructure by placing their network resources strategically at the network edges, providing a high-performance network for content delivering, namely Content Delivery Network (CDN). However, they offer an excellent service at a price suitable for big content providers, but excessive for small-medium ones.

The peer-to-peer (P2P) paradigm promises to solve the content distribution problem through a flexible application-level distribution, following the epidemic-style dissemination mechanisms where the stream is divided into small parts, called chunks, and exchanged among peers in a distributed fashion, without additional costs. The P2P technology has become increasingly popular to stream video contents, as shown by the success of several commercial systems such as PPLive[2], PPStream[3], UUSee[4], and SopCast[5].

Given this scenario, the big challenge is to design an efficient and reliable peer-to-peer content distribution system which enables users to enjoy their video contents anytime and anywhere.

1.1.1 Video Streaming Multicasting

Media streaming refers to a process of delivering multimedia content, such as video streaming, to a group of users called participants. The content is streamed regarding timing and quality constraints, and the main problem regards an efficient distribution process that satisfies the corresponding constraints and application requirements.

Basically, video streaming contents can be classified in two groups: on-demand and live. In on-demand video streaming, called Video on-Demand (VoD), contents are pre-recorded and stored in one or more servers, and their size is fixed and well known, making VoD similar to file transfer: the content can be partially buffered or completely downloaded before being watched, and timeliness constraints are not critical. Thus, the content can be played and paused by users, as well as rewound or fast-forwarded: users can navigate the content. For instance, users can download the episodes of their favorite TV Series at anytime, and watch them in the evening or during the weekend,

[2] www.pptv.com
[3] www.ppstream.com
[4] www.uusee.com
[5] www.sopcast.org

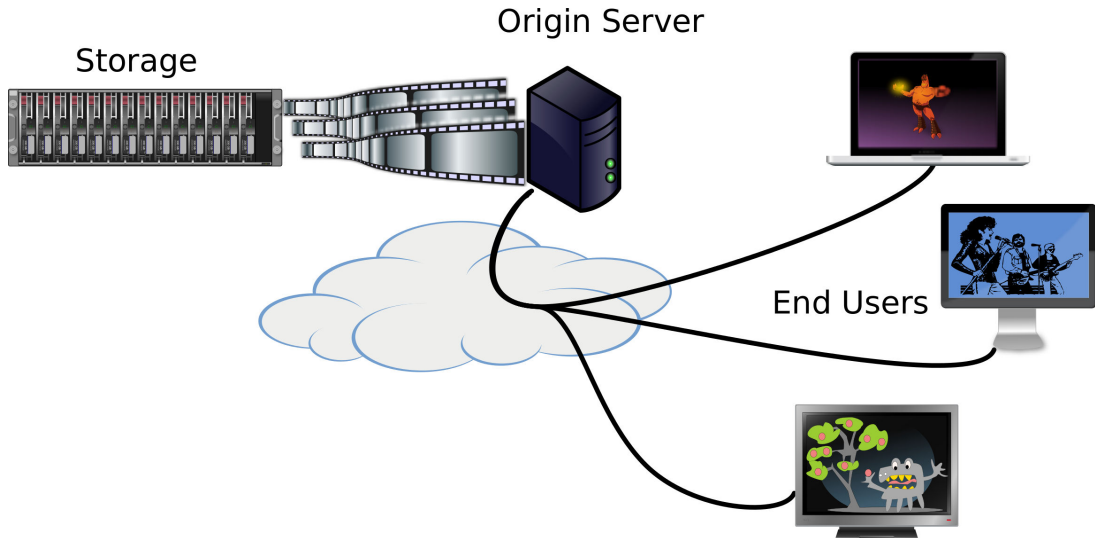

Figure 1.2: On-demand video streaming.

whenever they want. Netflix and Hulu are two companies that offer Video on-Demand both ad-supported and at monthly price.

In live video streaming, as represented in Fig. 1.3, the content to stream is generated during the execution of the corresponding event (e.g., football match), making the content available at a certain time for all the users, for instance when the football match starts, and cannot be fast-forwarded.

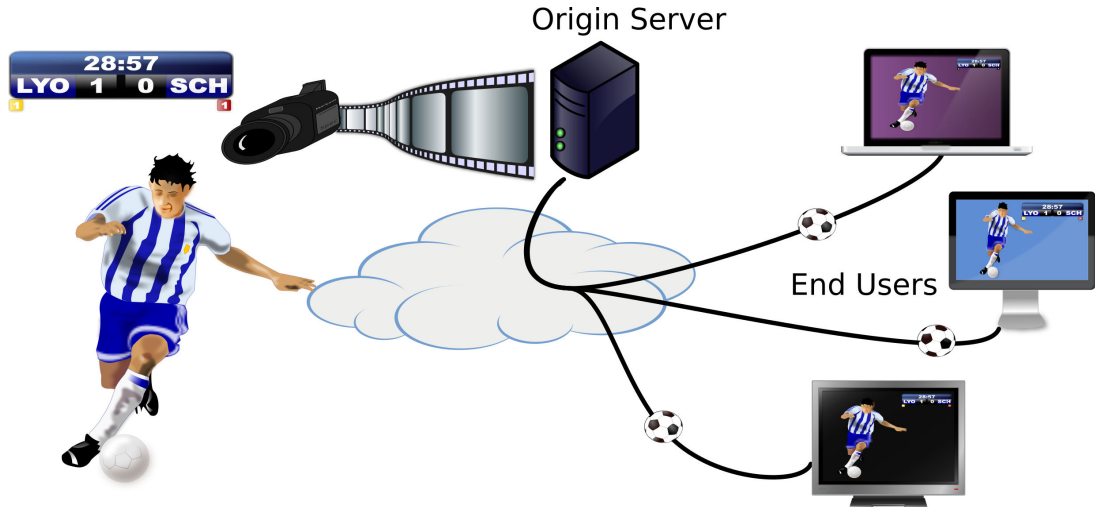

Figure 1.3: Live video streaming service.

Therefore, the size and the number of video packets that will be sent to participants are not known a priori, making the video distribution more challenging than in Video on-Demand. These characteristics emphasize the timeliness constraints of live streaming, affecting the quality experienced by customers: users are willing to watch the football match without interruptions, enjoying a sharp and clean live stream. Perhaps, they can wait a handful of seconds to establish the stream, buffering a few seconds, then, they are very demanding on its quality.

Live video streaming are delay sensitive services, reflecting the "play while downloading" model. The contents must be delivered within a limited delay, otherwise can be discarded, resulting in a wasting of time and resources. The source node can stream the content at different data rates and video formats, resulting in several channels, related either to the network resources or to the devices available at participants. For instance, TV broadcaster might distribute the same channel on regular and the High Definition (HD) quality, and the latter requires more resources.

Thus, there is a trade-off between the target video quality and the amount of resources employed.

1.1.2 IP Multicast

IP Multicast [80] provides a scalable point-to-multipoint delivery service for those applications that involve groups of users. IP Multicast delivers traffic from a source host towards multiple receivers, consuming less network resources than any other solution.

Basically, a multicast session is characterized by one or more nodes, called multicast sources, which inject multicast traffic, for instance video, towards a multicast group, flowing such packets through the multicast overlay: such packets are replicated by multicast routers towards participants, as showed in Fig. 1.4.

In particular, users announce their interest to join a multicast session to routers by subscribing to the corresponding multicast group address through the Internet Group Management Protocol (IGMPv3) [21]. After a router receives a membership subscription, it joins the corresponding multicast group address, being part of the multicast overlay. Then, it receives the data packets associated to the multicast group address, and it will forward such data packets towards the users interested in that multicast group. The multicast routing protocols enable only a subset of multicast routers to forward the multicast traffic towards the hosts or other routers, resulting in an overlay topology, usually a spanning tree, on top of the physical network.

Multicast routing protocols can be classified in two categories: source-based and core-based. In source-based multicast routing protocol, the source of the multicast session is the root of the multicast tree. Core-based multicast protocols construct a shared multicast tree for each multicast group, having a rendezvous point as the root of the tree. The multicast tree is shared among all the source nodes of the same group. Each source-node sends the multicast data to the core node that forwards such data to the shared tree.

There are several multicast routing protocol, they include Protocol Independent Multicast Sparse Mode (PIM-SM) [34], PIM Dense Mode (PIM-DM) [14], Distance Vector Multicast Routing Protocol (DVMRP) [102], Multicast Open Shortest Path First (MOSPF) [75], Border Gateway Multicast Protocol (BGMP) [59], Core-Based Trees (CBT) [15], and many other protocols.

IP Multicast is the most efficient solution for video streaming distribution: it employs the actual bandwidth needed to deliver the video packets, plus the signaling overhead introduced by the multicast routing protocol to establish and maintain the multicast overlay, and the data delivery delay corresponds to the minimum delay to reach all the receivers.

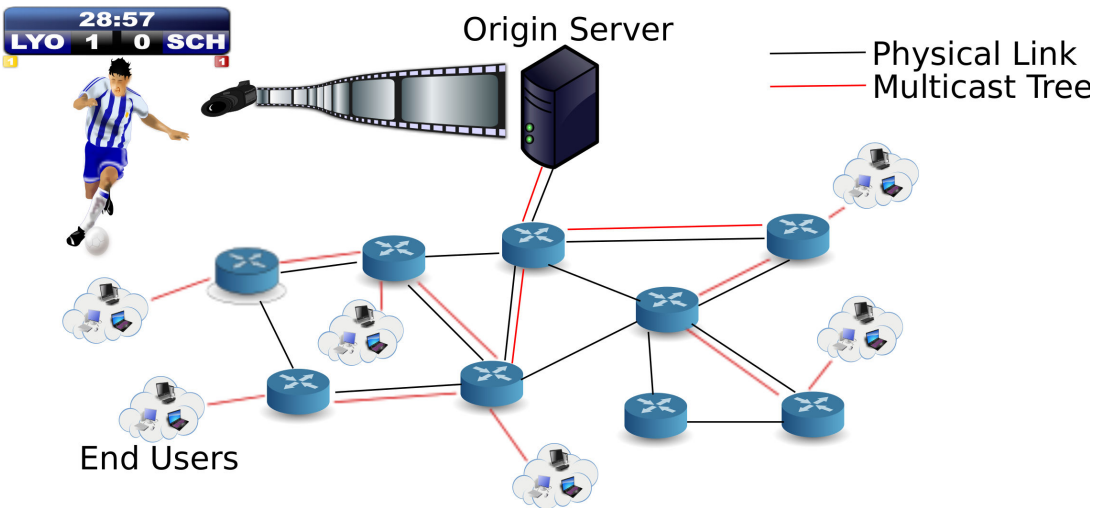

Figure 1.4: IP Multicast over a source-based tree topology.

However, IP Multicast suffers of several issues [33], they include assigning permanent and temporal multicast address, multicast data integrity mechanism, multicast access control, security and protection on multicast routes and sessions, group management, and sender authorization. In addition, several Internet Service Provides (ISPs) used to disable multicast in their core routers, or filter multicast traffic, or their backbone routers are too old to support multicast. Therefore, the IP Multicast service suffers of several issues and it is not yet available on the Internet.

1.1.3 Client-Server Architecture

The client-sever paradigm is the most common centralized computing architecture where servers and clients are connected through IP unicast connections, as depicted in Fig. 1.5.

Basically, each server provides a collection of services and the clients obtain the desired service through request messages. For instance, a server might offer a collection of video contents and the clients that are willing to watch them will issue the corresponding request messages.

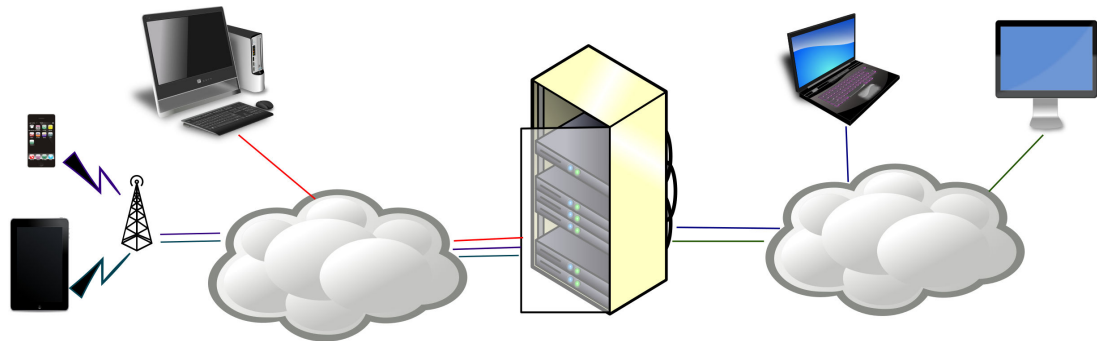

Figure 1.5: The client-server architecture.

Content providers might distribute their video contents through the client-server architecture, mainly because it is easy to set up and simple to maintain and to update. However, in order to guarantee the service availability and reliability, they have to reserve a noteworthy collection of resources from hardware resources to human resources, resulting in a significant investment of money.

In addition, the client-server architecture provides the access to only a limited number of clients, until the server's resources are saturated, limiting significantly the number of clients that will access to the video streaming service. Thus, the client-server model might be expensive to maintain and does not scale with the number of clients, being available to only a subset of clients, that is in contrast to the actual objectives of content providers.

1.1.4 Content Delivery Networks

A Content Delivery Network, such as Akamai, Weebo[6], and CoralCDN[7], consists of a collection of network and storage resources spanning the Internet. The origin server pushes the content over several servers that are placed strategically at the network edges, offering a transparent and efficient delivery to the clients, reducing significantly the latency at the end-users. This

[6] www.weebo.it
[7] www.coralcdn.org

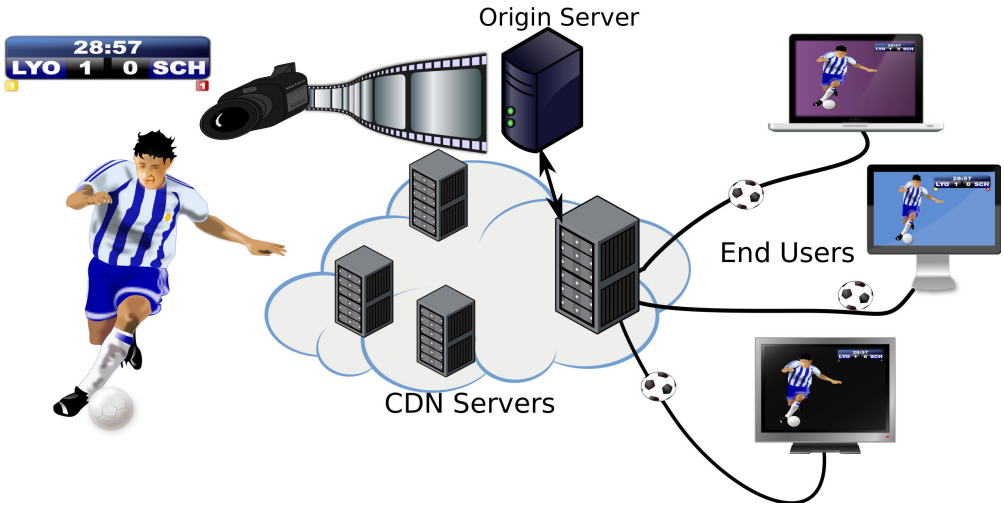

Figure 1.6: A content delivery network infrastructure.

architecture allows to distribute the burn of the clients traffic over several servers, reducing the load at the origin server and the network congestion, as well. Thus, CDNs overcome the limits of the client-server model by providing more servers, multiplying the amount of available resources, increasing the number of clients served [58].

Several content providers, such as USAToday[8] and New York Post[9], moved their Web sites to CDNs, obtaining fast, reliable, and scalable content delivery. Nowadays, video content providers and TV broadcasters, such as RAI[10], BBC[11], and CBC[12], leverage on CDNs to provide their contents, for instance full-length shows, movies, and TV series.

Although the CDN technology employs more resources than the client-server architecture, however, such resources are limited and might be insufficient to sustain the video streaming service, in terms of number of participants, number of video channels, and video quality data stream. In addition, small content provides, such as local TV broadcasters and radios, or indepen-

[8] www.usatoday.com
[9] www.nypost.com
[10] www.rai.tv
[11] www.bbc.com
[12] www.cbc.com

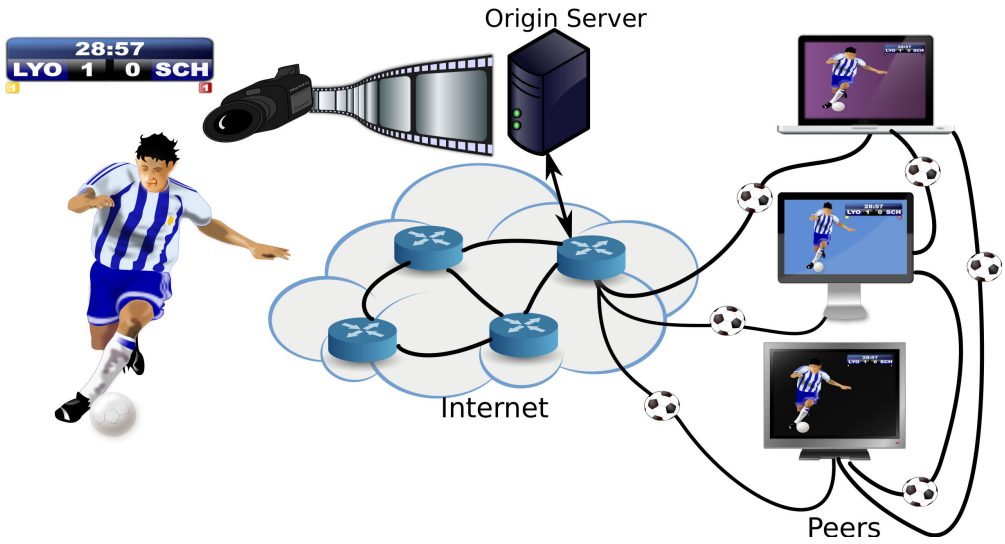

Figure 1.7: The peer-to-peer architecture.

dent filmmakers, cannot benefit of CDNs advantages because of their cost, that is not affordable for such small-medium content providers.

1.1.5 Peer-to-Peer Technology

The basic philosophy behind the peer-to-peer paradigm is to promote co-operation among end-users exploiting their resources: end-users behave as client and server at the same time, they are completely equivalent, namely peers. In particular, peers not only download data from the network, but become active entities of the distribution process by employing their bandwidth resources to upload the downloaded data to other peers, resulting in a significant load reduction at the origin server, as depicted in Fig. 1.7; Hence, the P2P paradigm addresses the resources bottleneck by employing the peers' ones.

The peer-to-peer technology owes its popularity to file sharing which aims to disseminate data files on the Internet in a distributed fashion. Today, emule[13] and BitTorrent[14] are the most used file sharing applications. Ac-

[13] www.emule-project.net
[14] www.bittorrent.com

tually, the P2P technology is employed in several services, including search engines (e.g., Faroo[15]), Voice over IP (e.g., Skype[16]), and other applications, representing a noteworthy part [86, 55] of the global Internet traffic.

Recently, P2P video streaming has emerged as a promising solution for Internet video distribution, as testified by the growing success of commercial streaming systems such as UUSee, PPLive, SopCast, and many others. Basically, P2P video streaming is characterized by one source node which transforms video flows into data packets that are injected into the P2P network. Then, peers establish several connections among them to exchange such data packets as well as additional information that can be helpful to improve the streaming experience.

1.2 Motivation of the Thesis

TV-like video streaming is a typical multicast application because involves a large number of participants that are willing to receive the same stream of video packets.

The actual content delivery can be realized in several ways. IP Multicast is the network solution which allows to deliver the same stream of packets to multiple receivers. Multicast packets are replicated by the network routers, without adding any additional load to the source node, using the least network resources needed. Unfortunately, the Internet Service Provides (ISPs) used to filter multicast traffic or to disable multicast protocols in Internet routers, making IP multicast unfeasible to distribute multicast traffic (e.g., video content) on the global Internet.

[15] www.faroo.com
[16] www.skype.com

The client-server paradigm is suitable to manage several permanent connections, emulating the IP Multicast through multiple IP unicast connections from the server to each client. However, when the population of clients increases, the number of IP unicast connections grows dramatically, and the server can either reduce the amount of resources (e.g., bandwidth) dedicated to each unicast connection (i.e., user), or reject clients connections, resulting in a quality degradation or a denial of service, respectively. The incapacity to scale with the number of clients and the limited amount of available resource at the server, make the client-server architecture unsuitable for delivering live streaming content to millions of end-users at any time.

The Content Delivery Networks (CDNs) overcome such limitations by deploying high speed array of servers, actually providing more resources than the client-server model, enabling the system to sustain more clients. However, content providers have to paying a fee for each GB delivered, which can be expensive for small-medium content providers.

In recent years, the Peer-to-Peer technology has emerged as a promising solution for Internet video distribution. This technology i) leverages the resources available at the end-users, revealing great potential to scale and ii) does not require any additional support from Internet routers and network infrastructure, resulting in a cost-effective and easy to deploy solution.

This thesis focuses on unstructured (i.e., mesh-based) P2P systems which exhibits better performance [71] than structured ones. The key component of such systems is the scheduler which i) realizes the Internet video distribution through either push-based or pull-based diffusion schemes, ii) combines different peer and piece selection algorithms, and iii) leverages on peers and network information, exploiting the overall system resources. Hence, the scheduler operates to satisfy the application timing and bandwidth constraints, through an efficient use of peers resources and information, optimizing the video quality experienced by end-users.

Given this scenario, the big challenge is to design an efficient and reliable mesh-based P2P streaming system, shedding an insightful light on the diffusion schemes and network parameters employed in video distribution, enabling end-users to experience their video contents anytime and anywhere.

1.3 Structure of the Thesis and Contributions

As outlined discussing the motivation, this thesis deals with protocols and scheduling strategies for unstructured P2P streaming systems, with particular attention and reference to large-scale, real-time applications like TV.

Chapter 2 reviews the state-of-the-art in peer-to-peer application-level multicast systems, describing what has been proposed so far in the literature to tackle the research problem described in Section 1.2.

Chapter 3 presents our research work, introducing a new class of hybrid protocols that alternates data dissemination via push with missed data recovery via pull, this contribution is summarized in Section 1.3.1.

Chapter 4 points out the issues that prevent the Protocol Independent Multicast protocol family to work on Wireless Mesh Networks, proposing a solution to tackle such issues and evaluating the Wireless-PIM protocol. Then, the hybrid protocols are applied to video multicast session in Wireless Mesh Networks, where the push is performed via standard multicast delivery, while the pull is used to recover missed data packets in a P2P fashion. This work is reviewed in Section 1.3.2.

Chapter 5 introduces the GRAPES library, a helpful toolkit for P2P multimedia systems, presenting the functionalities that GRAPES offers. GRAPES has been implemented in the NAPA-WINE project and it is currently the core library behind PeerStreamer, an open source P2P media streaming framework. This contribution is outlined in Section 1.3.3.

Finally, concluding remarks and insights about future possible research directions are provided in Chapter 6.

1.3.1 Hybrid Push/Pull Protocols in P2P Networks

This section focuses on a new class of protocols that combines push and pull diffusion schemes, called hybrid Push/Pull protocols. The class is known as mesh-based swarming Push/Pull systems or interleave protocols;

The first contribution examines both synchronous and asynchronous models, exploring the impact of protocols parameters, such as overlay connectivity type, peers bandwidth, and active neighborhood size, trying to identify the efficiency of such simple protocols under several scenarios, gaining insight to design the next protocol generation with performance and efficiency in mind. The result of our study shows that hybrid protocols can work without having additional information on the other peers, being suitable for high churn scenario, because a piece negotiation occurs before the actual transfer. Moreover, the overlay connectivity type (i.e., asymmetric or symmetric) has no influence when the neighborhood degree is greater than 12 peers. Furthermore, it is shown that the round-trip-time delay (RTT) is an important parameter to take into account for delay sensitive systems, and can be employed in peer selection algorithms to promote communication among closest neighbors, leading to a reduction of the chunks diffusion delay.

Therefore, this contribution will help to better understand content distribution systems and also make a contribution to the debate about the relative merits of basic, stateless schemes and status-based schemes. Furthermore, such a hybrid system shows sustainable and efficient streaming with resources limited to two times the stream rate, ensuring small overhead, a result that pure push or pull protocols achieve at the price of state exchange between peers: Such a class of protocols is suitable for delay tolerant content delivery

(i.e., file-based communications) as well as high-bandwidth delay sensitive media streaming, such as live video streaming and IPTV.

1.3.2 Cooperative Multicasting in Wireless Mesh Networks

The second contribution is divided in two parts. The first part analyzes the problem of extending the Internet multicast protocols to wireless mesh networks, in view of the third contribution. In particular, it focuses on the standard Protocol Independent Multicast (PIM) protocols family. Such a protocols family has been dismissed as non practical, or non feasible, assuming its straightforward application on wireless networks.

This contribution shows that the PIM standard based implementations improperly interact when employed across wireless mesh networks and identifies the issues that prevent the PIM protocols to work on such networks.

It proposes a solution which do not require the actual modifications of the standard, but require only minor modifications of the underlying protocol implementation, yielding to the Wireless-PIM-DM protocol version.

Then, an evaluation study is addressed through the Wireless-PIM-DM protocol implementation in a network modeled by using the ns-3 simulator program. Such a study investigates on the performance of a wireless mesh network with full static mesh nodes that serve a multicast session to either static or roaming users.

The performance results show that the Wireless-PIM-DM protocol works properly and efficiently across wireless mesh networks, and this protocol scales with the number of clients. In addition, a multicast video streaming session is run on PIM, showing that PIM achieves effective throughput and packet delay performance behavior.

The second part of this contribution introduces PullCast: A cooperative protocol for supporting multicast distribution. PullCast applies the hybrid

Push/Pull protocols previously explored in the Internet scenario to wireless mesh networks.

In particular, end-users receive the multicast data packets in *push*; Then, such users build their local neighborhood through hello messages, or any gossiping protocol, retrieving missed chunks via *pull* through unicast messages to 1-hop neighbors, exploiting spatial diversity and higher unicast data rates. The key idea is to complement the standard multicast delivery protocol (i.e., push), by enabling a peer-to-peer mechanism to recover (i.e., pull) a small fraction of missed data packets from nodes' neighborhood.

The results show a significant increase of the fraction of peers reaching the target quality, that is 95% of chunks, showing also a rather low chunks diffusion delay.

Finally, PullCast has been evaluated in highly structured mesh network, where the chunks recovery is limited to the equivalent of an 802.11 BSS, and in more complex scenarios, where the mesh nodes offer connectivity to clients in a seamless network using a single radio channel, showing that PullCast improves the system performances in both scenarios.

1.3.3 GRAPES: A Generic Library for P2P Streaming

The last contribution presents the GRAPES library. GRAPES (Generic Resource-Aware P2P Environment for Streaming) is an open-source library that provides basic functionalities for building P2P streaming applications. GRAPES aims at providing a working codebase that can be easily modified to experiment with and to integrate novel ideas, addressing the problem of rewriting the whole application for any solution to test.

Currently, GRAPES is the core library of PeerStreamer[17], an Open Source P2P Media Streaming Framework. Indeed, PeerStreamer is today one of the

[17] http://peerstreamer.org

most diffused Open Source systems for P2P TV and streaming applications and its use by researchers and practitioners is increasing steadily.

1.4 Topics Outside the Scope of the Thesis

There are important issues related to content distribution that are outside the scope of the thesis:

Overlay Discovery and Management. One of the main assumptions in P2P systems is that peers can easily get in touch with all the other peers involved in the same video streaming session. Indeed, such systems employ several mechanisms to discover the overlay network topology. For instance, gossiping protocols [48, 100] can be used to complement the overlay management protocol, discovering peers belonging to the same streaming session. Then, the overlay management protocol selects a subset of nodes to populate and update the neighborhood set to communicate with.

Cooperation and Incentives. The P2P paradigm works on the premise that peers cooperate among them, sharing their resources, in order to increase the actual system capacity [108]. However, selfish peers may be motivated to use more resources than those allowed, for instance by sending a large number of request for obtaining data packets, without contributing in the distribution process, actually pauperizing the overall system, or even exhausting the whole P2P streaming system. P2P video streaming systems should address such a problem, providing mechanism to identify and exclude such peers (e.g., removing them from their neighborhood). In addition, providing incentive mechanisms for video streaming distribution is more complex than file-sharing applications, because of the real-time nature of such application. The tit-for-tat [46] strategy has been successfully employed in

BitTorrent-like distributions and has been revised for video streaming applications [87].

Content Integrity and Pollution. The P2P paradigm assumes the peers honesty, however, it cannot guarantee such trustfulness. Peers may intentionally pollute the content by introducing errors or distributing modified data, yielding in a waste of time and resources to distribute wrong data packets [32]. A simple mechanism to avoid content pollution is to diffuse chunks digitally signed by the source node, allowing peers to decrypt the chunks signature, discarding wrong packets. In addition, peers may banish that malicious neighbors that have injected or sent polluted data, diffusing their identities in the system.

Network Coding. Network coding allows intermediate nodes to encode data packets in order to perform error correction, exploiting cooperation among nodes: a client should receive enough linearly independent combinations of packets to reconstruct the original content,[25]. For instance, wireless links (e.g., WiFi) suffer from packet loss due to signal attenuation and interference, and network coding can be employed to address such a problem over lossy networks.

Security and Privacy. P2P systems are vulnerable to attack by malicious or malfunctioning nodes at several levels [37]. Such nodes can i) alter the protocol functioning, for instance sending malformed, different, or false messages, ii) modify or remove some data or control packets, iii) steal other nodes information, for instance violating peers privacy by stealing information on the content they are watching, and iv) performing Denial of Service attack to some node or, even worse, toward the source node. In addition, streaming system should grant the access to only authorized

peers, guaranteeing that their information (e.g., peer's history) are properly collected, used, and maintained.

Wireless Physical Layer. Live streaming over wireless networks has more rigid bandwidth limitations and different network constraints than wireline networks. In particular, the wireless medium suffers of several issues including background noise, signal attenuation, multipath fading, shadowing, and interference, that make the wireless channel less reliable than wired medium. In addition, wireless devices are usually battery-powered and the energy consumption is a key parameter to define the corresponding battery life time, and thus, the up-time period of wireless devices.

Chapter 2

State-of-the-Art

2.1 P2P Live Streaming

Peer-to-peer networks are emerging as cooperative solution to provide scalable, reliable, flexible, and cheap to deploy multicast service at the application-level. P2P systems designed for live video streaming are very different than those designed for file sharing; Despite of file sharing systems, where the content is *open after being downloaded*, in live video streaming systems the content is *play while downloading*, introducing more requirements and constraints, where the main bottleneck is the uplink bandwidth. Such systems aim to deliver each piece of content within its deadline, through a clever resource management, avoiding content blackout, in order to provide high quality of experience to users.

P2P systems may be broadly classified as two-sided or one-sided, depending on whether peers exchange information or not. For instance, peers can exchange information about their content (e.g., buffer map), their resources (e.g., bandwidth), their neighborhood, and other information, at a price of introducing control messages overhead. However, there is a trade off between information exchange (e.g., frequency, type) and the corresponding benefits.

2.1.1 Principles

A peer-to-peer network consists of a group of nodes that communicate and cooperate among them to achieve a common interest, e.g., video streaming service. Such nodes are logically connected among them through virtual links, implementing a network abstraction over the underlying physical topology, namely overlay network. Hence, each peer maintains a list of peers that form its neighborhood: the overlay manager module is in charge of populating and updating such a list of neighbors. Therefore, peers establishes communications with their neighbors, exchanging video packets and information.

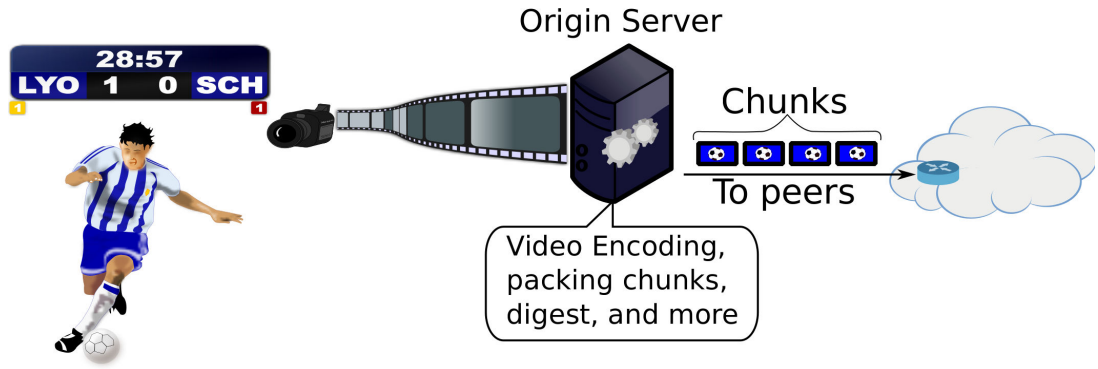

Figure 2.1: From video content to chunks.

Despite of peers, several entities can be involved in the P2P network. For instance, file sharing systems employ dedicated servers to provide the indexing services used to search files. Therefore, some services, critical or not, might be provided or supported through dedicated servers, which are easier to manage and update, while the data exchange is realized directly between peers. Such systems that include external entity, for instance servers, are called *hybrid* P2P systems, while *pure* P2P systems consists of only regular peers, without any external entities.

Basically, live streaming systems are characterized by one or more nodes that are in charge of distributing the media content, following the file swarm-

ing paradigm: the video stream is divided in piece of data of a limited size, called chunks. Hence, such nodes are called source nodes because they are the actual entry point of the media stream into the network. The source node encapsulates one or more chunks in IP packets that will be sent towards the P2P network, as depicted in Fig. 2.1. Afterwards, whenever a new chunk is generated, this fresh content is received by a few peers that will redistribute such information towards the other peers.

2.1.2 Overlay Topology

In a P2P network peers are logically connected among them. Such virtual connections are related to the underlying physical topology, resulting in an overlay network, as depicted in Fig. 2.2.

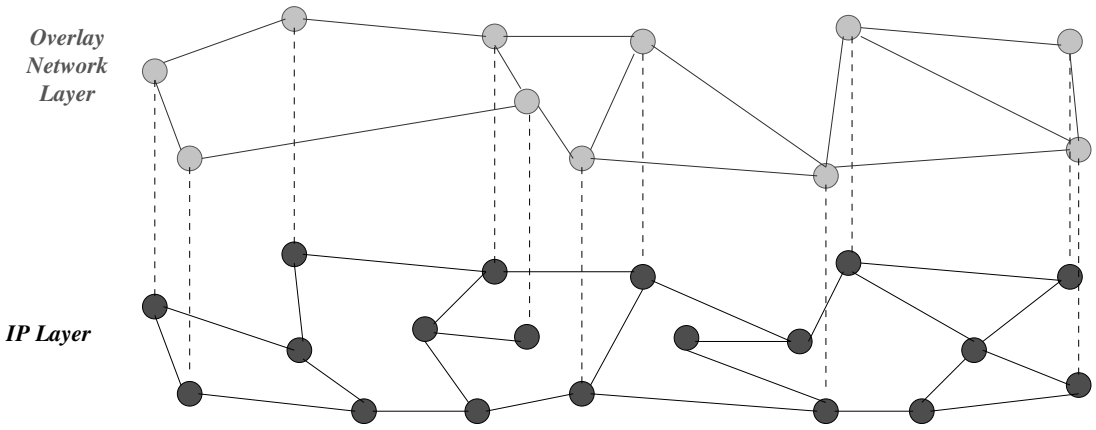

Figure 2.2: The overlay network layer.

P2P systems can be classified in structured and unstructured systems, depending on the resulting delivery tree. In particular, tree-based structured systems have the same delivery tree for all packets that corresponds to the actual tree overlay. On the other hand, unstructured overlay dynamically built the delivery tree for each single packet. while the packet travel into the overlay.

Structured Systems

In structured systems, the overlay management protocol entails a fundamental role, aim at building the tree structure multicast overlay, as shown in Fig. 2.3. The video source node is the root of the tree and peers establish parent-child relationships among them: each peer has exactly one parent node, while internal nodes become k degree parents, connecting to at most k children;

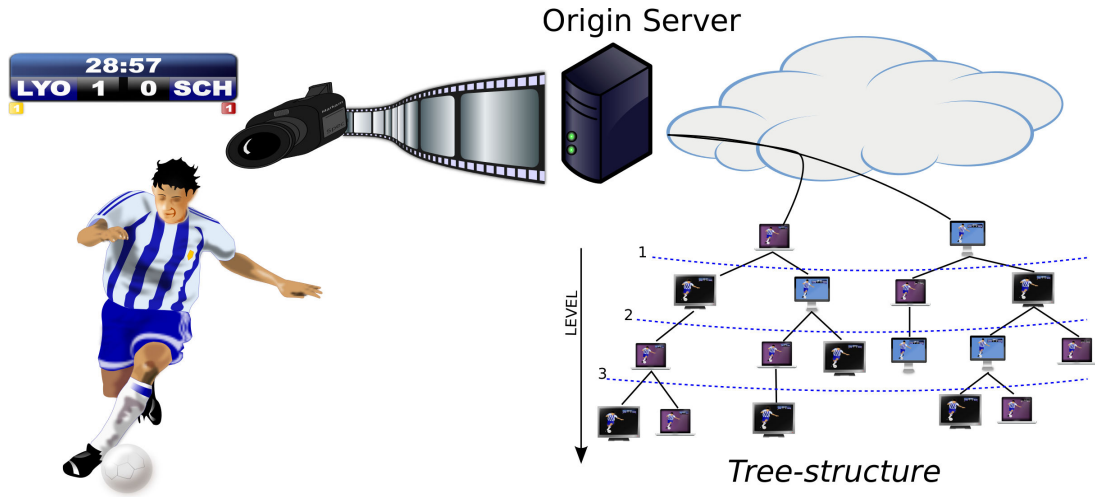

Figure 2.3: Tree-based overlay network.

Thus, whenever a peer receives a piece of data from its parent, it forwards such information to its children: the content flows through the tree structure, from the root to the leaves. Although the tree structure is straightforward and efficient, however, it suffers network dynamics significantly: the system must detect quickly such peers departure, otherwise all the peers in the subtree, rooted at the departed nodes, experience data starvation. Thus, the overlay management protocol has to take care of such issues, introducing signaling overhead: In structured systems the multicast tree overlay is the strong and the weak point at the same time.

Unstructured Systems

In unstructured systems peers are connected between them in a mesh topology, establishing neighborhood relationships, as show in Fig. 2.4. Such connections can be either symmetric or asymmetric, resulting in a directed or undirected graph, respectively. Mesh topologies are more resilient to peers departures than tree-based systems, because a peer departure results in an edge removed in the graph that can be replaced by adding another peer (i.e., edge) in the corresponding neighborhood.

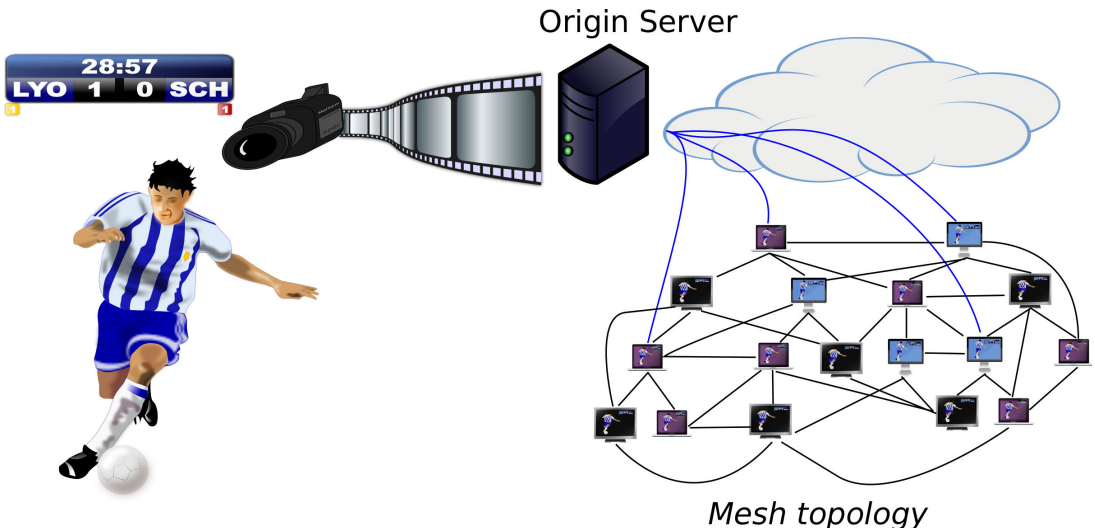

Figure 2.4: Mesh-based overlay network.

In such systems, each piece of data may follow a different distribution path, from the source node to each peer. The intelligence resides in the scheduler which use both neighborhood and network information, as well as its current state, to perform the content distribution. Basically, peers exchange several information among them, such as the fraction of chunks received or missed, the neighborhood size, the QoE score, and other information; The scheduler uses such information to select both the target peer to contact and the piece to exchange. In particular, the peer selection algorithm employs one or more policies to find a suitable peer in the neighborhood to contact, and the chunk selection algorithm chooses a set of chunks to exchange.

A comparative study between tree-based and mesh-based performance reveals [71] that swarm-like distribution in mesh-based overlay exhibits a superior performance. First, swarm-like distribution reduce the impact of low bandwidth peers. In case of tree-based overlay, the descendant peers of low bandwidth parents experience content bottleneck, affecting the corresponding sub-tree. Opposite is the case of mesh-based systems, where peers can fetch the video packets from alternatives paths, obtaining such packets from their neighboring nodes. In addition, mesh-based systems are more resilient to network dynamics than tree-based ones which require to re-organize the whole distribution overlay.

2.1.3 P2P Streaming Architecture

In a live streaming session, peers run the same P2P system, which consists of several modules. The NAPA-WINE project has defined an high-level system architecture [18], that include the user, the scheduler, the overlay, the monitoring, the information repository, and the messaging layer module. This architecture is shown in Fig. 2.5.

The relevance of each module is related to the overlay topology employed. For instance, in tree-based systems the overlay manager is the core module because it is in charge of establishing and maintaining the content distribution tree; In mesh-based systems the content distribution engine is the scheduler module. Thus, such modules interact among them, defining peers behavior and interactions, applying rules and algorithms to the content distribution.

User Module This module is in charge of receiving the video stream from the P2P application, decoding the stream and reproducing the content to the user's device.

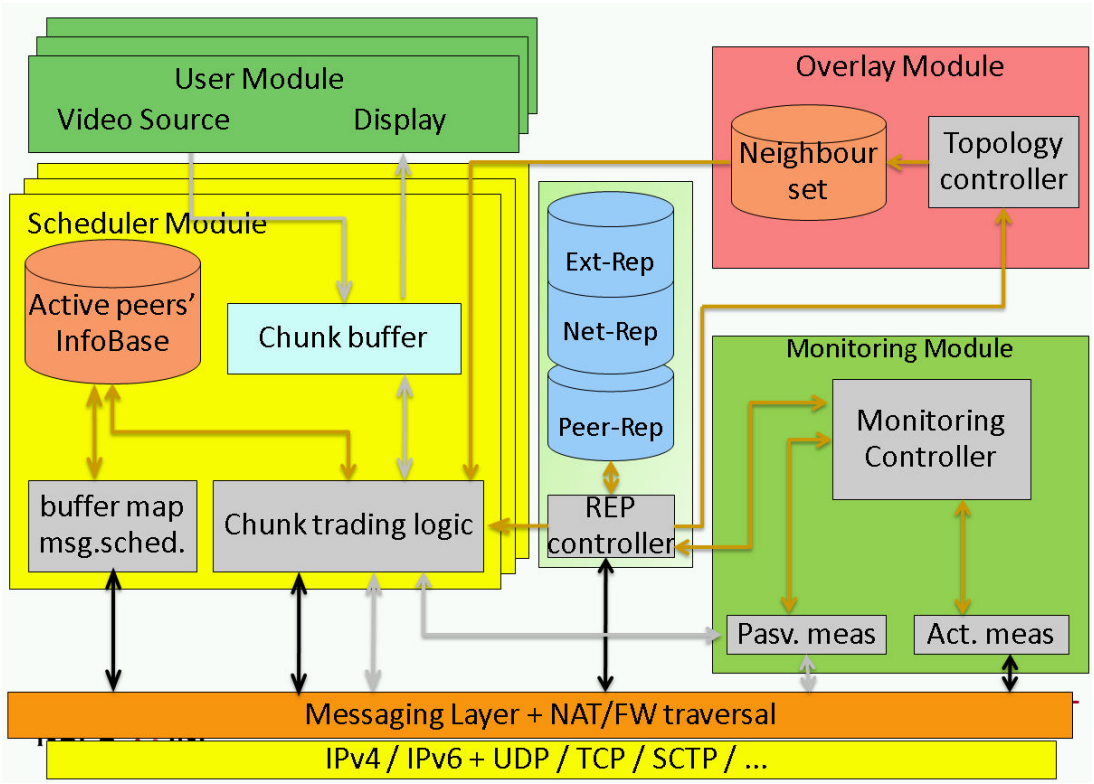

Figure 2.5: The NAPA-WINE architecture.

Furthermore, the user modules might provide information about the streaming quality experienced by the users.

Monitoring Module P2P systems often fail to notice the importance of the underlying network conditions, the dynamic countermeasure made by ISPs to tackle such traffic, and the possibility to combine such network measurements to design a network-aware systems.

This module is in charge of performing measurements on the peer's environment and resources. It provides information on the network access (e.g., NAT), resources (e.g., bandwidth) and latency (e.g., RTT) between the local peer and its neighbors, and other metrics. The importance and the impact of such measurements to the content distribution system is is one of the main achievements [18] of the NAPA-WINE project.

Overlay Module The main task of this module to create and maintain the peer Neighborhood, which populates this set with a large number neighbors, guaranteeing resilience and flexibility to overlay dynamics and session switch.

Moreover, each streaming session has its corresponding overlay network; Thus, whenever the peer switches from one streaming session to another one, this module has to rebuild the local peer Neighborhood for the new streaming session.

Repository Management Information are very helpful to perform wise operations. This module stores information and statistics about peers status, environments, and resources in a centralized database.

Such information can be used by several entities. For instance, peers can enhance their neighborhood, or improve the session switching mechanism, network providers can tune the resources allocated to the P2P network, and content providers can evaluate the impact of their contents.

Scheduler Module The scheduler is the core of the distribution process in mesh-based systems. It interacts with the other entities of the system, integrating the information gathered from the other peers, with network parameters and measurements, as well as the information provided by the overlay manager, and the current peer state.

The scheduler is in charge of executing the peer and chunk selection algorithms to select the chunks to exchange with the other peers. The algorithms can be classified in two classes, depending on whether the system is push- or pull-based, namely diffusion schemes. Push-based systems are *sender* oriented, because the sender manages the actual content distribution, deciding the chunks to diffuse through such algorithms. On the other hand, pull-based systems are receiver oriented, because the receiver selects the content to receive.

Furthermore, the order of executing such algorithms provide another classification, based on whether the chunk is selected before the peer, or vice versa, stressing the diffusion more on the chunks or on the peers, respectively.

2.1.4 Requirements and Trade-off

This section describes the main requirements of P2P live streaming systems and the corresponding trade-offs.

Startup and Buffering Delay. P2P systems usually underestimate the start-up phase. Whenever a peer join the networks, the overlay-discovery operation is done, collecting information about other peers. Then, the peer receives data packets from the P2P network, establishing connections to exchange such packets, and, after a short time, the playout at the peer starts. Thus, the peer experiences i) the join and overlay-discovery delay, becoming part of the P2P network and ii) the buffering delay, collecting enough chunks to play the video. Some resources can be dedicated to minimize such delays, increasing the quality perceived by users. Similar observations can be applied for channel switching.

Scalability. P2P systems are supposed to scale with the number of participants. Even an unexpected increase of participants, also with heterogeneous resources, should be manage by the system, at a price of marginal effects. Commercial live streaming systems (e.g., PPLive, CoolStreaming, SopCast) show how big is the actual P2P population, reporting that the size, depending on the video content, is between hundreds of thousands and millions of users, and such a population is growing faster, because of the increasing penetration of broadband access and the entertainment offers.

Overlay Management and Reliability. P2P systems employ several protocols, such as gossiping protocols, discovering a partial view of the overlay network topology. Such protocols can be employed to identify changes in the network topology. In particular, peers can leave the network at any time, in graceful or ungraceful (e.g., failure) ways. The dynamics of the network, characterized by peers that continuously join and leave the network is called churn. Whenever a peer leaves the network, it brings its resources and the received content away from the network, possibly leaving obsolete information about its actual presence into the network. Thus, P2P system has to address the churn problem, limiting its impact on the overall system, possibly promoting connections among stable peers, limiting the effects of network dynamics on video quality experienced by users.

Quality of Experience. Quality of Experience (QoE) describes the quality of multimedia and video services. The most common metrics [20, 68] used to estimate the QoE are the chunk delivery delays, the playback rate, the fraction of chunks received, as well as the peak signal-to-noise (PSNR) ratio. The latter measures the distortion between the received video image and the original one. However, QoE is influenced by additional factors [103], such as the users quality expectation and users attention.

2.2 State-of-the-Art Solutions

So far, the literature proposes several works to address the live streaming problem. Indeed, unstructured P2P systems are difficult to analyze, because of their i) dynamic interaction among peers imposed, ii) real-time requirements, and iii) the unpredictable delivery path taken by each chunk according to the peer and chunk selection algorithms. Recent traffic measurement studies [41, 28] on commercial P2P streaming systems show how such systems

are aggressive in consuming peers resources, especially bandwidth. Hence, several techniques can be employed in content distribution systems, leveraging either on the network topology and the information exchanged between peers, to optimize the resources utilization (e.g., bandwidth).

The following sections surveyed the state-of-the-art of P2P streaming systems that have been proposed in the literature as follows. Section 2.2.1 reviews the tree-based systems while Section 2.2.2, instead, describes mesh-based ones, and Section 2.2.3 discusses the combination of tree-based and mesh-based systems, namely hybrid systems. Finally, Section 2.2.4 presents the NAPA-WINE architecture.

2.2.1 Tree-based P2P Streaming Systems

DONeT [111] (Data-Driven Overlay Network) organizes nodes in a dynamic tree, depending on data availability. DONeT consists of the *membership manager*, to preserve a partial knowledge of the overlay network and exchanging buffer map among peers, the *partnership manager*, to maintain collaboration with other active nodes, and the *scheduler*, to schedule data transmissions towards suitable suppliers. Although such an adaptive behavior, i.e., nodes selects best partners, leads to nodes clustering, which depends on their buffer maps, it may lead to unbalanced-tree that affects end-to-end delay while the buffer map exchange increases the overhead in the network.

CoolStreaming is the commercial implementation of DONet. In [110] the authors presents an empirical study on CoolStreaming, investigating on users' behavior, their correlation, and the perceived video quality; Next, the authors redesigned [105] the system, introducing a pull-push mechanism, adopting sub-streams, multi-source, and multi path delivery, to reduce the content diffusion delay, and introducing multiple servers to boost join procedure.

Finally, they investigate [64] on users network access (e.g., firewall, NAT), finding out that users contribution are significantly influenced by their net-

work access. In addition, they collect and analyze real traces of live event, investigating on several system parameters, such as the packet block size, number of sub-streams, peers uplink capacity, partner set size, and start-up time.

ZIGZAG [92] is a single source tree-based protocol, which guarantees a balanced tree. ZIGZAG employs an administrative module to separate the maintenance and the data layer. Thus, whenever a parent left the data delivery tree, the administrative part recoveries such a departure, handling ungraceful node failure, at a price of protocol complexity and signaling overhead.

PROMISE [39] is a video streaming implementation on top of the P2P application-level service CollectCast [40]. CollectCast gathers information of the underlying network topology, such as bandwidth variation, end-to-end delay, and peers failure. In PROMISE a session is established whenever a peer requests a content: first the receiver performs a look-up request to the underlying network that provides a collection of peers that can supply the requested content. Then, it constructs the topology and fills active and standby set, connecting active candidates to the receiver that establishes parallel connections towards all such active senders, using the standby set as a backup.

SplitStream [25] is a multi-tree system that splits content in several stripes, e.g., k stripes, delivering each stripe on a separate tree. Peers join as many tree as there are stripes they wish to receive. SplitStream uses the multi-tree topology to exploit leaves' bandwidth: each node is an internal node only in one tree, being a leaf in the others. SplitStream employs MDC where each stripe is a description and a parent node failure implies the loss of at most one description. However, such a protocol is built on top of a tree structure, suffering of all the issues related to tree topology: maintenance, reliability, node failure detection, and recovery.

CoopNet [97] was originally proposed as a web-cache system to tackle the problem of flash crowds at web sites. A flash crowd is an unexpected surge

of traffic which overload the queried server. When a flash crowd occurs, the bandwidth is the main bottleneck, and when the server provides dynamic web pages, the CPU becomes another bottleneck. The same authors proposed a revised version of CoopNet [98] for distributing media content among peers. The original architecture has been converted in an hybrid-P2P system where the server provides contents to peers and maintains also a list of peers served and contents provided to them. When the server is going to be overloaded by peers requests, it redirects such peers to those that have already received the corresponding content before: Such a mechanism produces a set of tree rooted at the source, where peers forward contents to the others, using Multiple Description Coding (MDC) schemes to reduce the churning effects.

ChunkySpread [99] is a multi-tree system which leverages on frequent signaling messages to dynamically changing parents node, in order to fulfill load balancing and reducing latency delay. Furthermore, it provides a loop detection mechanism based on bloom filters.

2.2.2 Mesh-based P2P Streaming Systems

In [81] authors propose a queue-based scheduler that employs queue techniques from the router world. The scheduling algorithm adapts peers' behavior to the state of their corresponding queues. Each peer holds a playback buffer which stores the chunks received and a forwarding queue for the content to forward: the content received is classified either to forward (chunks received as a reply to a pull message) or not (received from other peers), and whenever the forwarding queue is empty a pull message is issued to the source that fills the forwarding queue. Although such a system achieves full bandwidth utilization, however, it uses several noteworthy advantages: (i) full mesh network, (ii) the server is able to send a chunk to all peers in the network at the same time, (iii) the signal propagation delay is negligible, (iv) data delivery is performed quickly because the chunk is made up by a small

amount of data, (v) signaling messages does not suffer any queuing delay and is processed immediately. In addition, the source server corresponds to the bootstrap node, thus, churn phases may overwhelmed the source, increasing the pull response time.

So far, several mesh-based systems which combines scheduling policies and overlay management protocols have been proposed in the literature [67, 69, 73, 36].

In [29] authors investigate on the performance improvements achieved by exploiting information on peers' upload bandwidth in the overlay topology construction, as well as the peer selection, leading to a reduction of the the chunk distribution delay.

A new chunks trading scheme based on the offer-select messages has been proposed in [24]. Actually, the offer-select mechanism is a three-phase protocol: i) each peer advertises the chunks it possesses through an offer message towards neighboring nodes, ii) the latter reply with a select message which contains the subset of chunks they are willing to receive, and iii) finally, the original peer collects a sequence of selected chunks in their outbound queue, which are sent and Ack-ed, and before the transmission queue empties, a new offer-select cycle is started. The combination of dynamic chunks trading and transmission queue enable nodes to utilizing the upload bandwidth efficiently, improving the overall system performance in terms of delay and lost video packets.

A comparative study [20], analyzes several peer/chunk selection algorithms in push-based systems, assuming that each peer knows everything about the others, including the transmissions they are going to initiate in the next future. The authors prove the optimal rate diffusion achieved by the combination of random peer selection and latest useful chunk algorithms, while random peer selection and latest blind chunk algorithms plus coding schemes at the source achieve delay optimality.

Another comparative study [66] focuses on both simple and sophisticated algorithms, evaluating five different protocols on the source rate, the bandwidth usage, buffering techniques, and other metrics. They start from a simple random pull protocol which avoids wasting transmission due to chunk duplicates, introducing a pull-token mechanism to publish bandwidth availability, so peers may reply either pulling a sequence of pieces or notifying that they have the same chunks. The authors compare simple and complex schedulers, showing that streaming quality is not scheduling-dependent in presence of low streaming rate and are wide playback delays; On the other side, by increasing the streaming rate, the delay constraints become too strict, and the intelligence of the scheduler is fundamental. Finally, they also observe that systems which guarantee high streaming rate with low delay are too complex and need some particular conditions (e.g., full-mesh overlay), and the source's bandwidth is crucial in such systems because increases chunks diffusion noteworthy as well as rich peers (e.g., good heterogeneity) that have more bandwidth than the others.

Pulse [79] uses a pull technique and builds a general mesh topology with a complex neighborhood structure for each node, trying to favor a distribution of nodes such that more endowed nodes are closer to the source and help an efficient distribution.

2.2.3 Hybrid P2P Streaming Systems

The near-optimality of simple-pull based protocol has been demonstrated in [109] through simulation and real-world experiments. Such optimality is paid with a source bandwidth above several times the streaming rate and exchanging buffer maps, increasing significantly the signaling overhead. There is a question about when reporting such information, because there is a trade-off between delay and signaling overhead. The authors deal with this trade-off by proposing a pull-push protocol using the sub-streams approach; whenever

a peer successfully receives a packet belonging to a given sub-stream via pull, it sends a sub-stream subscription, asking the target peer to directly push the chunks of that sub-stream. The pull mechanism is used when a packet expected in push is not received within its deadline. This protocol builds a push-tree for each sub-stream, with all the problems related to the tree structure, thus, nodes departures may cause starvation and pull overhead to recover from such failures.

In Prime [70] participants form a randomly connected and directed mesh where each peer has multiple parents and child peers. The parents are assigned randomly and incoming and outgoing connections of each peer have different path in order to reduce the probability of a shared bottleneck among them. Receivers "drive" the distribution, which means that also in Prime the basic mechanism is pulling information from other nodes. This system produces a significant amount of traffic due for reporting periodically new chunks received, and during churn phases the bootstrapping node may be overwhelmed.

2.2.4 The NAPA-WINE Architecture

The NAPA-WINE[1] (Network Aware Peer-to-peer Application over WIse NEtwork) project proposes an innovative architecture [18] for High Quality P2P live streaming applications [63] over unstructured networks. Such an architecture promotes cooperation between the application level and the underlying network layer to offer high quality streaming service to users, limiting the impact of the P2P application on the underlying network. This solution introduces the monitoring module to collect real time information on the network conditions and users' video experience. Such information are helpful to improve the actual content distribution, and can be used locally by each peer, to reconfigure the scheduling policies, to modify the overlay

[1] www.napa-wine.eu

topology, or by the source node to tune the content distribution. In addition, such information can be stored in a shared repository that can be used by the network providers (i.e., ISPs) to improve the P2P distribution. Indeed, the NAPA-WINE project did not limit its work to only propose such an architecture, it actually implements PeerStreamer:[2] an open source P2P media streaming framework.

[2] www.peerstreamer.org

Chapter 3

Understanding Hybrid P2P Push/Pull Protocols

3.1 Introduction

The rise of Peer-to-Peer (P2P) communication paradigm seems to be the next big thing in Internet services. After the initial phase, where legal concerns about illegal exchanged content hampered the diffusion, the key idea of sharing resources to improve the common benefit of users is gaining momentum, and service providers are modifying their business plans and strategies to ride the rising tide.

Multimedia applications are dominating the Internet in terms of bandwidth usage (at least if we include in this traffic also the download of video files). IPTV to VoIP and video conference applications are among the fastest growing applications, and those based on P2P communications are starting to dominate the lot (e.g., PPLive, SopCast, CoolStreaming, Skype). The main goals in the design of a video streaming system are i) provisioning information delivery within given time bounds, and ii) make an efficient use of the available bandwidth. In addition the system should be simple and resilient to churn and overlay dynamism, as well as scaling well with the number of users and the length of the stream.

Traditional approaches for peer-to-peer streaming systems propose streaming on either structured or unstructured overlays. The former are characterized by parent/children relationships over a tree-structure, where the information flow [25] through the multicast tree, pushed from parents to children. On the other side, in unstructured overlays peers are organized in mesh topology where they either push [79, 70] to or pull [109] information from their neighbors, depending on the diffusion scheme and the algorithms employed. Among these systems, those based on swarm-like distribution of the chunks over unstructured overlays are the most successful, (see for instance commercial applications like PPlive, SopCast, UUsee, TVAnts, etc., or scientific literature [93, 70]). Early performance works [23, 22, 17, 19] propose new semi-analytic techniques to explore fundamental properties of the topology used by P2P overlays as well as the protocols used to build them. Indeed, some recent works [71, 20, 29] use the push scheme in unstructured overlays, that have superior performance than structured ones, hinting to the fact that the structure of the overlay and the information distribution process are not necessarily coupled.

Two recent works [69, 85] proposed a system called Interleave where nodes are synchronized and alternate regularly push and pull phases where push is used to disseminate fresh content to target nodes while pull is used to retrieve missing information. The proposal is appealing because of good asymptotic performances, and because it reduces almost to zero the need for signaling. However, all the analysis is based on asymptotic assumptions and synchronized, cycle-based operation.

There are few other works that propose a combination of pull and push [109, 69]. Most of them use the push mechanism for spreading rapidly the content, and the pull mechanisms for filling the holes in the received stream or to subscribe to different trees in case of a multiple-tree structure. In any case, no proposal considers to *interleave* the pull and push mechanisms.

Interleave was designed mainly for P2P file transfer. The analysis of the scheme made in [85] was done considering that all the nodes in the network are homogeneous and synchronized, i.e. the download time of a single piece is the same for all nodes and the time is slotted. Authors of [85] found that the file dissemination time of Interleave is within a constant factor of the optimal performance of a fully centralized system. A major concern is whether the good performance of Interleave is due to synchronization and bandwidth homogeneity or not. Also, while [85] speculates that Interleave may be used for live streaming, it does not further investigate this issue.

In this work, we translate the synchronous scheme into a fully distributed asynchronous protocol, where nodes in the network behave independently, without any coordination, actually introducing a new class of asynchronous push-pull protocols for live video streaming systems. Thus, we compare synchronous and asynchronous models, and evaluate the impact of protocols parameters, such as the dimension of the active neighborhood, and the influence of topologies, such as symmetric or asymmetric topologies, identifying the efficiency of such very simple protocols in different scenarios, gaining insight to design the next protocol generation with performance and efficiency in mind. Then, we focus on understanding the interactions between push and pull phases, we explore the importance of parallel download and possible choice on chunks to either push or pull. Finally, we investigate on the impact of upload bandwidth, the role of push and pull diffusion schemes, and the importance of network parameters, in particular, the round-trip-time, evaluating a peers selection algorithm that promote communications between closest neighbors.

The results show that the protocol we propose is able to maintain the performance of the synchronized Interleave scheme in case of file distribution; moreover, it is able to provide delay bounds in case of live streaming. To the best of our knowledge it has never been shown that a scheme as simple as

Interleave, which is a combination of pull and push and it does not maintain information about the pieces owned by nodes, can already provide such a good performance. The notation used in this chapter is reported in Table A.1.

The remaining part of this chapter is organized as follows. Section 3.2 describes the fundamental workout of push/pull protocols, both in a simplified, cycle-based model and in a more realistic, entirely distributed system. Section 3.3 discusses numerical results and fundamental system parameters. Section 3.4 provides some observations on fundamental study of push/pull protocols. Section 3.5 introduces the chunks negotiation mechanism and parallel transmission. Section 3.6 describes the performance metrics. The results are discussed in Section 3.7. Section 3.8 reports several comments on the protocol. Finally, Section 3.9 describes a delay-aware protocol to promotes communications among closest peers. The evaluation metrics are reported in Section 3.11 while the experimental setup is described in Section 3.12, and Section 3.13 presents the obtained results. Section 3.14 presents the conclusions and describes future directions.

3.2 Interleave Protocols

We consider a system with a single source, where the content is partitioned into pieces that can be exchanged independently. Pieces are generated at a constant rate B_s, which can be the streaming rate or simply a service rate for a file transfer. Each piece has a sequence number that reflects the order of creation by the source. Each node *alternates* between pull and push mode, and has a finite size neighborhood $\mathcal{N}$ defined by its contact list of size $|\mathcal{N}|$; the neighborhood of peer $\mathcal{K}_j$ is defined as the set of peers that can be contacted *actively* by peer $\mathcal{K}_j$, i.e., $\mathcal{K}_j$ can contact peers only in its contact list, but can be contacted by peers that are outside its contact list.[1]

[1]This work was supported by the Italian Ministry of University and Research (MiUR), with the Grant PRIN-2006099023 “Profiles” (disi.unitn.it/profiles)

In *push mode*, a peer $\mathcal{K}_k$ randomly selects a neighbor and a piece to push; if the neighbor does not have the piece and has available download bandwidth, $\mathcal{K}_k$ uploads the piece, otherwise the push is aborted.

In *pull mode*, a peer $\mathcal{K}_k$ randomly selects a neighbor and a piece and sends a pull request to that neighbor. If the neighbor has the requested piece and it is currently not uploading to any other node, it accepts the request, otherwise it refuses it.

The piece selection policy represents a delicate part of the design, especially for streaming systems, where each piece should be received within a maximum delay. A random piece selection may work for file distribution, but not for live streaming. Also, 'intelligent' selection procedures, like rarest-first and similar, cannot be implemented without state signaling between peers, thus cannot be implemented in a basic push/pull protocol without state. We adopt the same piece selection policy proposed in Interleave [85]:

- In push mode, the node pushes the piece with the highest sequence number among the pieces received *via a push* from one of its neighbors.
- In pull mode, the node asks for the piece with the *lowest sequence number it does not possess*. This policy aims to fill the holes within the sequence of pieces.

Interesting features to be explored in the future, include simple choice-based selection policies and strategies to maintain the system without state, thus highly dynamic and resistant to churn.

3.2.1 A Cycle-based Model

The basic scheme described above was initially proposed considering a simplified environment, where all the nodes have the same upload link

Part of this work was published in the proceedings of the Second International Conference on Communications and Electronics (ICCE 2008) Hanoi, Vietnam [2].

bandwidth and unlimited download link bandwidth. It is also assumed that the time to upload a single piece is much bigger than the latency for the exchange of a single (application level) message. Additionally it is supposed that the nodes are synchronized and the time slotted. In even slots all the nodes are in push mode; in odd slots all the nodes are in pull mode.

If the node is in push (pull) mode and the request is accepted by the selected neighbor, the piece is pushed (pulled). At the end of the slot the node switches mode. If the request is refused by the selected neighbor, the slot is entirely wasted, and the node waits until the end of the slot before switching mode (and sending a new request).

The source pushes a new piece in every even slot, and it replies to pull in odd slots. This means that the streaming rate is equal to one piece every two cycles and the required upload bandwidth is $2B_s$.

This cycle-based model was employed in [85] to find theoretical bounds for the distribution process, and we use it for comparison reasons. In a realistic scenario, upload link bandwidths are heterogeneous, downlink capacity is not infinite, and imposing synchronization may be impractical (if at all possible), thus a better model is required to understand fundamental properties.

3.2.2 A Realistic Model

In a real distributed system, the behavior of each node is independent from all other nodes. Nodes are desynchronized, because forcing synchronization is costly and can also lead to high inefficiency (synchronization can be based only on the least performing peer). A node switches from one mode to the other (e.g., from push to pull) either after a request is accepted and the corresponding piece transfer is finished, or after receiving a maximum number of refusals to requests (at each try, a new neighbor is randomly selected). This behavior keeps the system running smoothly and avoid starvation and blocking.

The time spent in pull or push mode depends on the success of the requests and on the availability of the bandwidth (both, the neighbors' and the node's bandwidth); thus each push or pull interval is not fixed and the nodes, even in a homogeneous case, will be desynchronized.

We stress that this implementation is entirely decentralized and the information exchange among peers is kept to a minimal level: Keep-alive messages for the contact list management and push/pull queries and answers. Additionally, the goal of this work is not building a new system (not yet at least!), but to gain insight into fundamental parameters like the contact list size, the efficiency of the distribution system (i.e., the ratio between the available bandwidth in upload/download and the streaming or transfer rate). Algorithms 1 and 2 summarize the basic operations of a single node related to *sending out requests*; Algorithms 3 summarizes the basic operations related to replying to requests. We do not report the pseudo-code for building and maintaining the contact list for the sake of brevity.

Notice that these simple algorithms generate asynchronous behaviors even within the same node. Indeed, a node controls its own alternating between push and pull in sending out requests, but has no control on the requests he receives, thus a node can be in one of four states, depending on its *active/-passive* status (not considering periods of idle behavior during the signaling related to requests' negotiation): Pushing/pulled, pulling/pushed, pushing/-pushed and pulling/pulled. We use the desinenceing to identify the active status consequent to sending out a request, and the desinenceed to identify the passive status of answering a request. Notice that in the two states pushing/pulled and pulling/pushed, the two coexisting transmissions compete for the same uplink (downlink respectively) resources.

Algorithm 1 Interleave: Sending Out Requests in **PULL** state.

```
1: Input: maxPullAttempts
2: while (pullAttempt < maxPullAttempts) do
3:     n = Select a neighbor //Select a neighbor to Pull
4:     c = Select the lowest piece not owned //Select a chunk to recover
5:     SendPull(n,c) //Pull chunk c to node n
6:     Wait for reply: Run Pull timeout GoTo TimeoutExpired
7:     if (reply == AcceptPull) then
8:         if (has download bandwidth available) then
9:             SendReady(n,c) message
10:            Wait for receiving piece: Run timeout GoTo TimeoutExpired
11:            StartPullingPiece
12:            break //exit from the loop cycle
13:        else
14:            SendBusy(n) message //Node's downlink is saturated
15:        end if
16:    end if
17:    TimeoutExpired:
18:    pullAttempt++
19: end while
20: status = PUSH
21: pullAttempt = 0 //Reset attempts for the next cycle
```

3.3 Numerical Results

We have implemented both the cycle-based and the more realistic, asynchronous and distributed model in a simulation environment, to explore some fundamental parameters.

The cycle-based implementation works in the same ideal hypotheses of the theoretical analysis in [85] as summarized in Section 3.2.1.

The asynchronous model is based on the protocol described in Section 3.2.2, along with Algorithms 1 and 2 and 3. It include the signaling for pieces exchange and some simple models of the underlying information transport network. The actual transmission of pieces, e.g., with TCP/IP, is

Algorithm 2 Interleave: Sending Out Requests in **PUSH** state.

```
1: Input: maxPushAttempts
2: while (pushAttempt < maxPushAttempts) do
3:     n = Select a neighbor //Select a neighbor to Push
4:     c = Select the highest piece owned //Select a chunk to transmit
5:     SendPush(n,c) message //Push chunk c to node n
6:     Wait for reply: Run Push timeout GoTo TimeoutExpired
7:     if (reply == AcceptPush) then
8:         if (has upload bandwidth available) then
9:             SendReady(n,c) message
10:            StartPushingPiece
11:            break //exit from the while cycle
12:        else
13:            SendBusy(n) message //Node's uplink is saturated
14:        end if
15:    end if
16:    TimeoutExpired:
17:    pushAttempt++
18: end while
19: status = PULL
20: pushAttempt = 0 //Reset attempts for the next cycle
```

not implemented since is would slow down simulation and make the results so complex as to make their interpretation almost impossible.

3.3.1 The PeerSim Environment

PeerSim is a Java based simulator that consists of many configurable components: It has two types of engines, *cycle-based* and *event-based*, and different modules that manage the overlay building process and the transport characteristics. In the *cycle-based* engine, all nodes are synchronized, making it a perfect tool for the implementation of the simple cycle-based model. In each cycle each node is activated sequentially and executes a protocol: There is no concurrency among nodes nor competition for resources. In the *event-based* engine, all nodes are independent and run concurrently (they are indepen-

Algorithm 3 Interleave: Replying to Requests.

```
1: if (message == PULL) then
2:     if (has piece c && has upload bandwidth available) then
3:         SendAcceptPull(n,c) message
4:         Wait for reply: Run Reply timeout GoTo TimeoutExpired
5:         if (reply == READY) then
6:             StartPieceUploading
7:         end if
8:     else
9:         SendRefusePull (n,c) message
10:    end if
11: end if
12: if (message == PUSH) then
13:    if (pieceID missing && has download bandwidth available) then
14:        SendAcceptPush(n,c) message
15:        Wait for reply: Run Reply timeout GoTo TimeoutExpired
16:        if (reply == READY) then
17:            StartPieceDownload
18:        else
19:            SendRefusePush(n,c) message
20:        end if
21:    end if
22: end if
23: TimeoutExpired:
```

dent instances of the same class). They can add events in the event list of the simulator, so nodes can compete for resources.

With PeerSim it is possible to build different network overlays; the models for some of them (e.g., random graph) are present in the simulator. It is also possible to specify if the edges of the overlay graph are directed or not. With random graphs, the degree of each node is random, because it depends on how many incoming edges have been created. We create a model with constant-degree, where all the nodes has the same number of edges (and neighbors). This overlay is similar in spirit to the one built by BitTorrent.

Parameter	Value
B_{UP}	128, 256, 512 kbit/s
$\|\mathcal{K}\|$	10^2, $5 \cdot 10^2$, 10^3
$\|\mathcal{C}\|$	10^3, $5 \cdot 10^3$, 10^4

Table 3.1: Parameters used for exploring Interleave protocols.

The simulator contains a transport layer for sending messages from a source to a destination: The layer adds a delay uniformly distributed between a minimum and a maximum value an it can also drop messages with a probability p.

For a detailed description of PeerSim simulator the interested reader is referred to [74]. The software modules developed in this chapter are available at the author's home page or on request.

3.3.2 Parameters under Study and Settings

The theoretical analysis of the basic Interleave scheme in [85] is done considering a directed random graph with degree $|\mathcal{K}| = k$. Each peer selects its own k neighbors independently from any other peers and without constraints. Each peer has a contact list of exactly k neighbors, and the probability of being in the contact list of other nodes is binomially distributed with mean k. We refer to this overlay as *asymmetric*.

We also define and test another overlay model, where the edges are not directed: In this case we use a constant-degree graph with degree k. Building it is somewhat more complex, since the choice of neighbors to build the contact list, requires signaling and coordination and can be difficult if the contact list is large and the number of peers small. We refer to this overlay as *symmetric*.

The source injects a piece per second, resulting in a the streaming rate B_s equal to the chunk size. In the cycle-based model this implies that each

cycle is equal to 0.5 s. In the event-based case, we set the piece size to 15.625 kB (125 kbit). So, the minimum bandwidth assumed for the uplink (128 kbit/s) is sufficient to leave some capacity for maintenance and piece request messaging.

We start considering a homogeneous case, where all nodes have the same bandwidth, and we test two different homogeneous bandwidths: 256 kbit/s and 512 kbit/s. Note that, with a bandwidth of 256 kbit/s, the source can serve each piece twice, because the streaming rate is one piece per second. Thus, this case corresponds to the cycle-based case, where a new piece is generated every even cycle and served again in odd cycle. If not otherwise stated, each node has a limited download bandwidth equal to four times the upload bandwidth.

We analyzed many scenarios with different network sizes and number of pieces. In particular, the number of nodes $|\mathcal{K}|$ can be equal to 100, 500 and 1000. The number of pieces $|\mathcal{C}|$ can be equal to 10^3, $5 \cdot 10^3$ and 10^4.

In order to evaluate the performance of different models with different parameter settings, we consider three main performance indexes:

The diffusion delay of a chunk is

$$\delta_p(c) = T_p(c) - \overrightarrow{T_{\mathcal{S}}}(c)$$

where $T_p(c)$ is the time in which peer p receives chunk c, and $\overrightarrow{T_{\mathcal{S}}}(c)$ is the time the source generated chunk c. $\overline{\Delta}$ is the chunk diffusion delay averaged over multiple realizations of the simulation.

Completion time In case of file distribution, the main performance metric is the time at which all nodes receive all pieces.

Maximum diffusion delay In case of live streaming, the delay of each piece represent the performance index of primary interest. We compute it considering all the nodes and all the pieces. Piece are numbered $0, \ldots, |\mathcal{C}|$. Piece

c is injected by the source at time $\overrightarrow{T_S}(c)$, and it reaches the node $\mathcal{K}_j$ at time $T_j(c)$. The maximum diffusion delay δ_{max} is the maximum over all pieces and over all nodes:

$$\delta_{max} = \max_{c,j}(T_j(c) - \overrightarrow{T_S}(c)) \quad c \in \mathcal{C},\ j \in \mathcal{K}\ .$$

Number of operations Since each node alternates between push and pull, it is important to understand how much efficient are these two mechanisms; we compute the number of pieces retrieved via pull and the number of pieces received via push.

For the different scenarios, we perform 20 independent realizations and we compute the Empirical Cumulative Distribution Function (ECDF or simply CDF) by combining the realizations.

3.3.3 Fundamental Cycle-Based Properties

In this Section we present the result for the cycle-based system. The purpose of this Section is twofold: to validate our implementation against the results presented in [85] and to show other performance measures, in order to gain more insight into the operations.

For the validation, we consider the asymmetric overlay with various degree k. The results presented in [85] focused on the completion time and they showed that, for $k > 8$, it reaches a stable value approximately equal to $2|\mathcal{C}| + 2\log|\mathcal{K}|$.

In Fig. 3.1 we show the full CDF of the completion time for different values of the contact list size (degree) k and for different values of number of pieces $|\mathcal{C}|$. In order to be able to compare the results for different $|\mathcal{C}|$, we show the completion time as the number cycles *after* the source has pushed the last piece. We observe that the completion time becomes stable for $k > 12$ (the upper group of curves; lower curves refer to $k = 8$), a value that is slightly

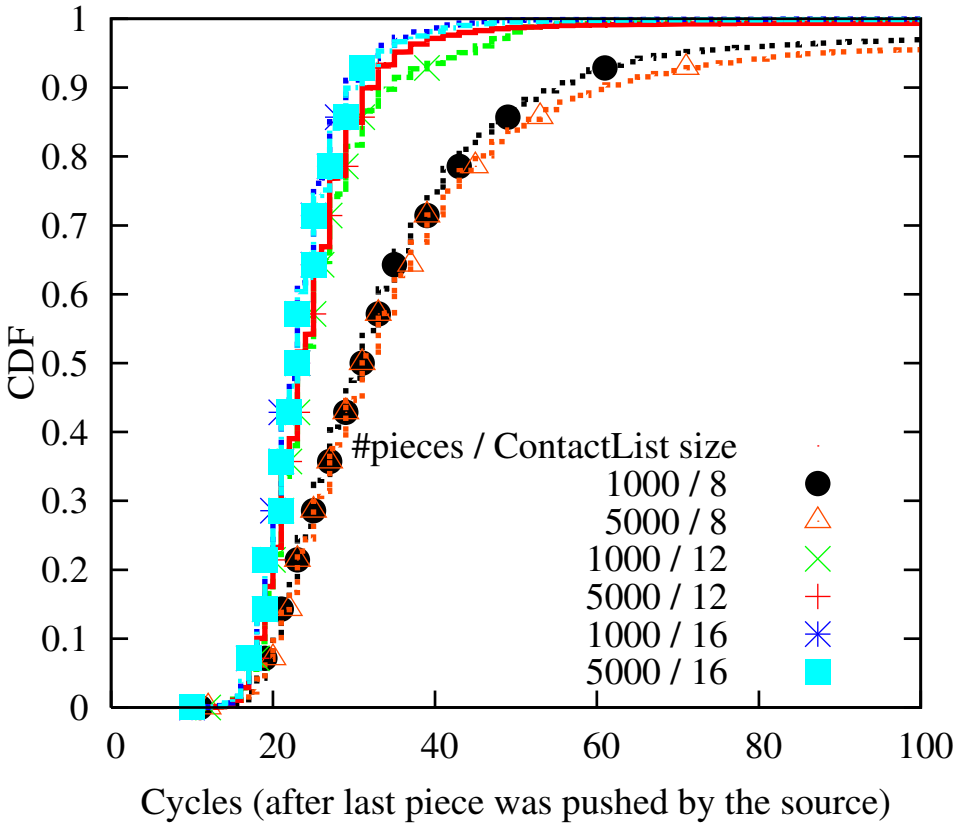

Figure 3.1: CDF of the overall download time with $|\mathcal{K}| = 1000$ nodes: Asymmetric contact list.

higher than the one presented in [85]. Since there are not many details in the simulation methodology used in [85], we are not able to investigate the reason of these differences. In any case, the qualitative behavior, that the contact list can be kept small without jeopardizing the results, remains the same.

The performances in case of too small contact list size k are due do the difference in the number of incoming edges. The nodes responsible for the tail of the CDF are the ones that are poorly connected. In fact, in a random graph with mean degree $k = 8$ there is a non-negligible probability that nodes are only in one or two contact lists, or even in none, so that most pieces must be pulled actively, since the probability of being pushed is small. This problem is reduced if we increase the mean number of contacts. We found similar results for different overlay network sizes (not shown here for space constraints).

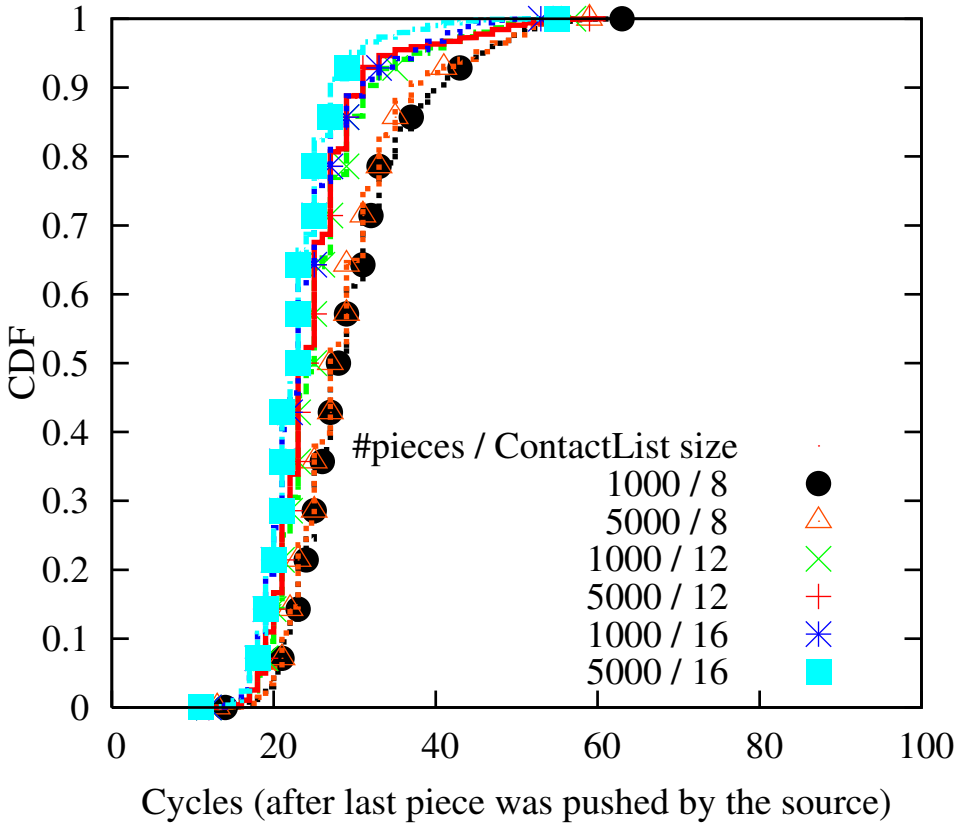

Figure 3.2: CDF of the overall download time with $|\mathcal{K}| = 1000$ nodes: Symmetric contact list.

In case of symmetric overlay, each node has a constant number of contacts and the problem observed in the asymmetric case disappears. Fig. 3.2 shows the CDF of the download time in case of symmetric contact list.

Comparing the CDF of the completion times in the different scenarios, we observe that the performances are almost equivalent, independently from the type of overlay (symmetric and asymmetric, provided that the degree is greater than 12), or from the number of pieces. This is valid for any performance measure we consider. For this reason hereinafter we will show only results where the contact list is symmetric and the degree k is set to 16.

The evaluation in [85] focused on the completion time, since Interleave was proposed for file distribution. Nevertheless, the piece selection policy

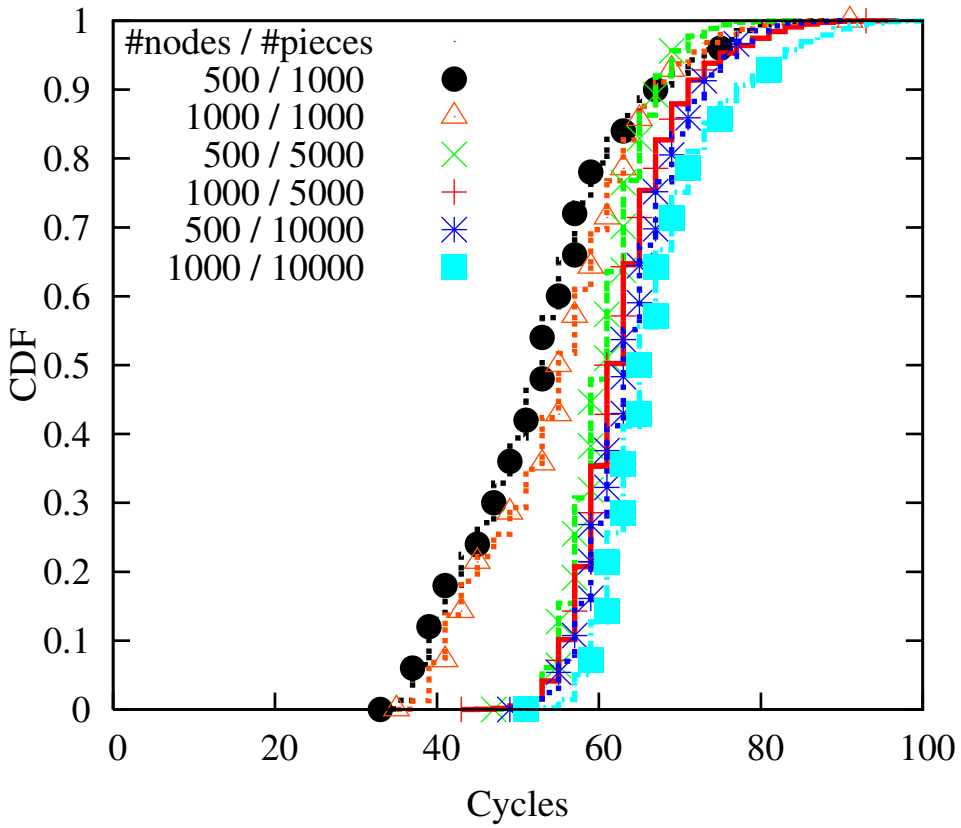

Figure 3.3: CDF of the maximum delay: Symmetric contact list, $k = 16$.

used by the scheme takes into account the order of creation of the pieces. Hence, it is interesting to study if the protocol can be used for distributing a stream. Fig. 3.3 shows the CDF of δ_{max}, the maximum delay. The maximum delay is slightly influenced by the number of pieces $\mathcal{C}$, especially the maximum value of the CDF. This means that, by properly setting the initial delay, the streaming can be received at the application level without interruption.

An interesting future work will address the analysis of delay percentiles. Using Forward Error Correction or Network Coding techniques, it is sufficient to receive a percentage f of the information to reconstruct the whole flow. Analyzing the delay behavior as a function of the f-th percentile will give indications on the design of the multi-coding technique to be used, as well as on the amount of redundancy needed to support a streaming with a given maximum delay.

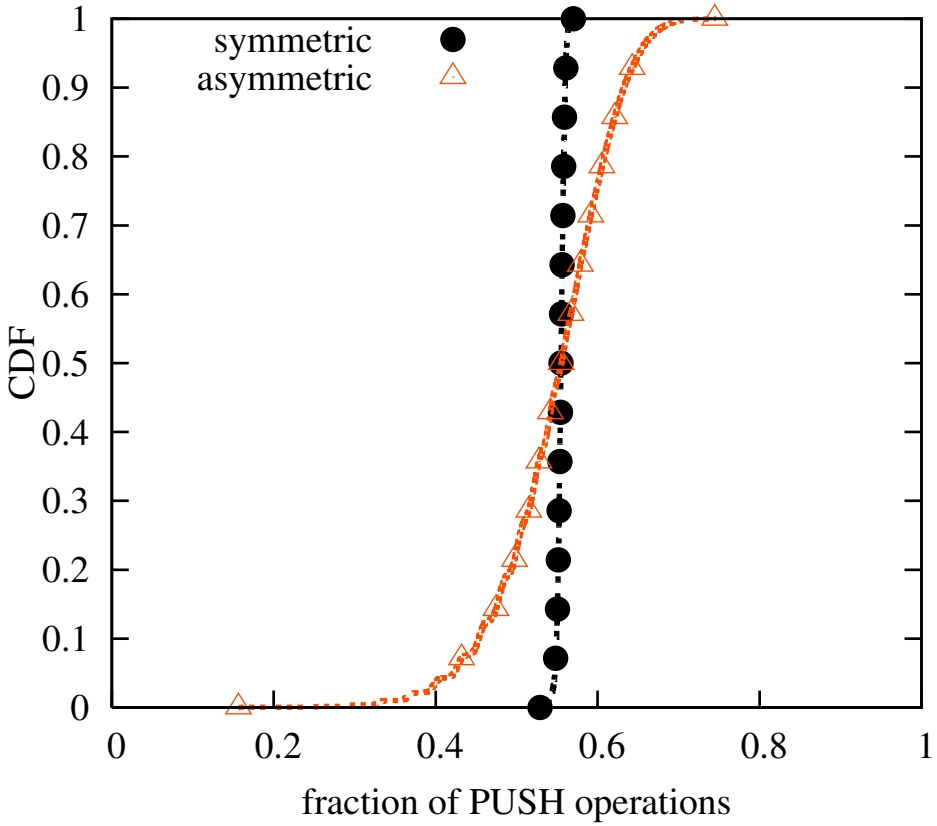

Figure 3.4: CDF of the Push operations: Symmetric contact list, $k = 16$, $|\mathcal{K}| = 1000$, $|\mathcal{C}| = 10^4$.

We notice that the maximum delay is expressed in number of cycles, thus finding efficient ways to reduce the piece size, and consequently the cycle duration, is an efficient way of supporting live streaming. Reducing the piece size to a single video frame (25 ms) will reduce the maximum delay to less than 2 s which is already a good value for streaming purposes.

Finally, in Fig. 3.4 we show the number of pieces received by push for different overlay types, asymmetric and symmetric. In both cases more pieces are received by push then by pull. In case of asymmetric overlay, the distribution has greater variance, since the overlay connectivity distribution has greater variance. The symmetric case shows instead almost perfect balance over all nodes, with slightly more pieces received via push.

3.3.4 Evaluating a Realistic Model

While the cycle-based is interesting as a first step in the analysis, the realistic model is able to show if the scheme still maintains the good performances in a scenario with no synchronization (and possibly heterogeneous). We start comparing the completion time obtained from the cycle-based model with the results obtained with the realistic model. To this aim, in the cycle-based model we set the duration of the cycle equal to 0.5 s, while in the realistic model we set the upload bandwidth equal to 256 kbit/s.

Fig. 3.5 shows the CDF of the completion time with the two models: Not only the realistic model protocol is able to work without synchronization, but it obtains also better performances in term of download time. For the realistic model, we consider also different scenarios: Also in this case, the overlay connectivity type (asymmetric or symmetric) has no influence if the degree is greater than 12. Moreover, different number of pieces $|\mathcal{C}|$ and number of nodes $|\mathcal{K}|$ give similar results.

The factor that mainly influences the performance here is the upload bandwidth of the nodes (including the source). We considered the case with upload bandwidth close to B_s, the rate of the streaming (i.e., 128 kbit/s): In this case the system is unstable and it is not possible to reach convergence.

Fig. 3.6 shows the CDF of the maximum delay for two different upload bandwidths. The impact of number of nodes on the CDF, especially with an upload bandwidth equal to 256 kbit/s, is mainly due to the greater number of hops that pieces have to do to reach all the nodes. An interesting result is represented by the decrease of the maximum delay as the upload bandwidth increases. The initial delay for a streaming application that uses the Interleave protocol is only few pieces.

Analyzing the delay, it is interesting to investigate the scalability in terms of number of pieces and number of nodes. To this aim, we consider the max-

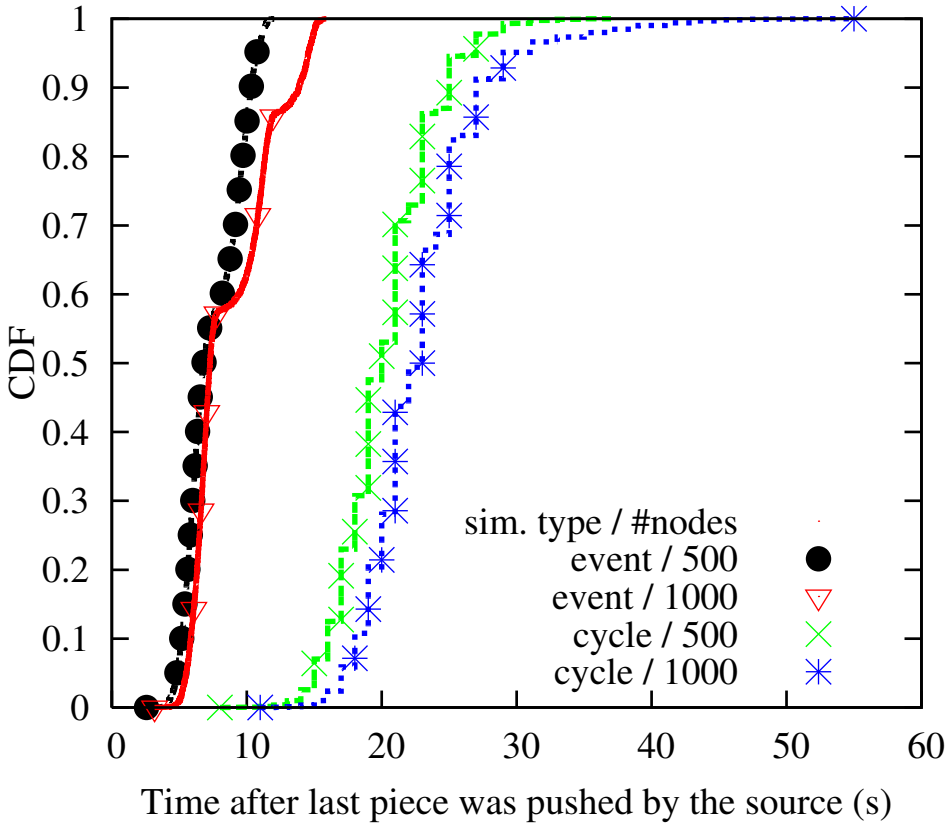

Figure 3.5: CDF of the overall download time: Symmetric contact list, $k = 16$, $|\mathcal{C}| = 5 \cdot 10^3$, $B_{UP} = 256\,\text{kbit/s}$.

imum value of the CDF of the maximum delay, and we compare different scenarios. Fig. 3.7 shows how the delay varies as the number of pieces increases. Especially when the bandwidth is equal to twice the streaming rate, there is a non negligible increase in the maximum delay. This means that the scheme may not be able to sustain long streaming and different mechanisms that are able to maintain the delay bounded are necessary.

3.4 Remarks

We considered a class of protocols suitable for supporting both file-based and stream-based communications in P2P overlay networks.

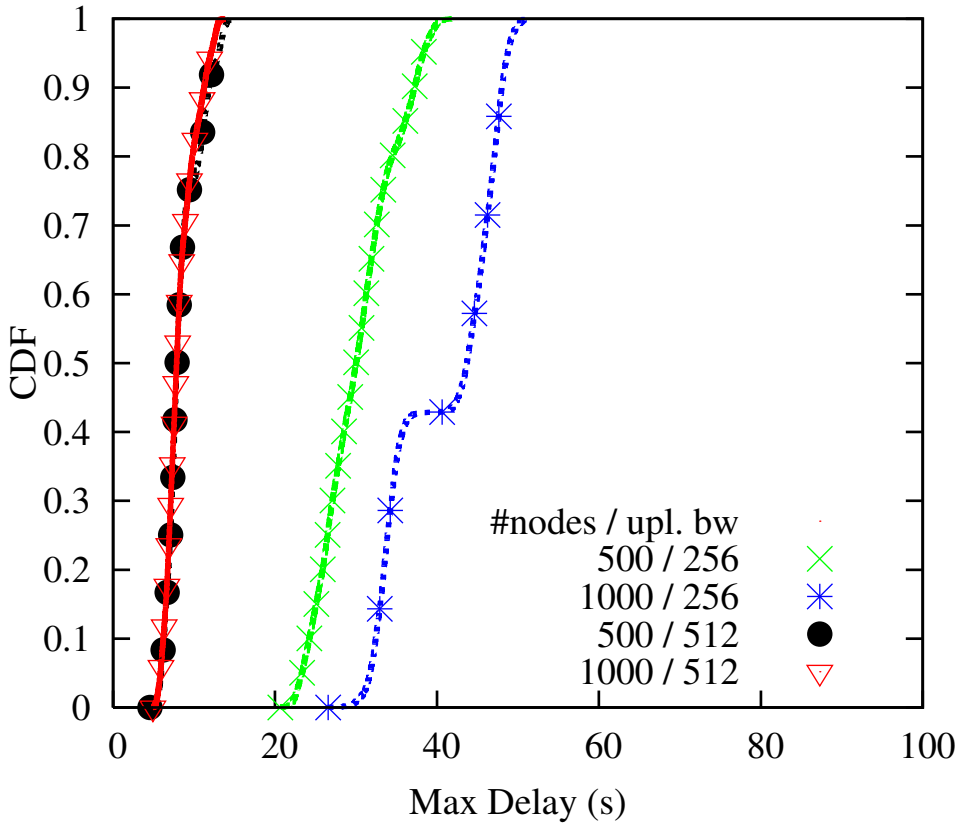

Figure 3.6: CDF of the maximum delay: Symmetric contact list, $k = 16$, $|\mathcal{C}| = 5 \cdot 10^3$.

The advantage of the protocol we analyze is that it can work without making any assumption on the node behavior: The interaction between two nodes is limited to the exchange of a single piece, making it suitable to situations with high churn. We evaluate the performance of this basic scheme, considering that it provides a lower bound on the performance achievable by any system, since adding other mechanisms, for instance, spreading the information about pieces the nodes own, should normally improve the performance.

These results are helpful to understand hybrid content distribution systems and also make a contribution to the discussion about the relative merits of basic, stateless schemes and status-based schemes.

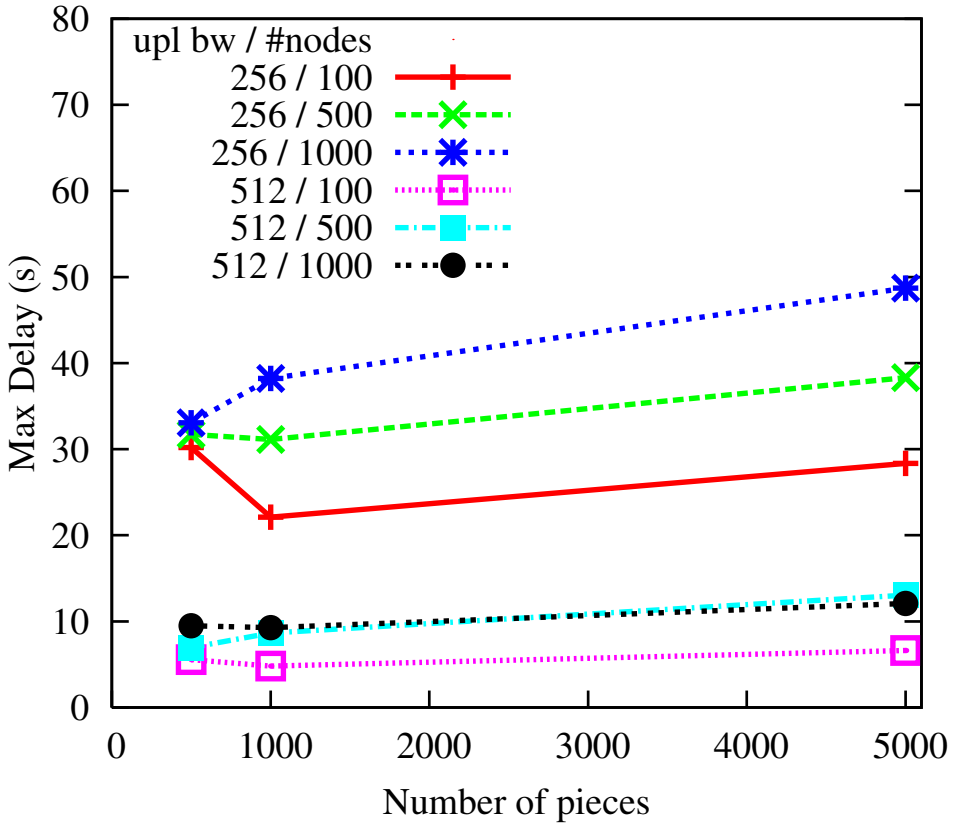

Figure 3.7: Maximum value of the maximum delay: Symmetric contact list, $k = 16$.

3.5 Enhancing the Push/Pull Protocols

We focus on the impact enabling peers to negotiate the chunks to transfer in both push and pull phases and to pursue parallel transmissions, investigating on the impact of peers bandwidth.[2] We assume that peers have no information about the status of neighbors (limiting signaling overhead), so that also in presence of high churn we expect the performance will be almost unaffected. In other words, we propose and evaluate a system with elementary characteristics only, so that the performance obtained can be used as a benchmark (hopefully a lower bound) for the performance of more sophisti-

[2]This work is supported by the European Commission through the NAPA-WINE Project (www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant No. 214412.

Part of this work was published in the proceedings of the INFOCOM Student Workshop 2009, Rio de Janeiro, Brazil [4].

cated systems based on the same principle.

The source is a normal peer with the same resources of the other nodes. This is very important since other studies assume that the source is a super-peer with more upload bandwidth, and in general, more resources than other nodes. This assumption affects (indeed improves) performance results non marginally: a source that has n times the bandwidth needed for the stream, i.e., B_s, can inject the information n times in the system, practically "seeding" n parallel distribution processes. We analyze the impact of both upload and download bandwidth on streaming performance, using a priority-sharing bandwidth management system. Focusing on streaming, the simple idea of pushing recent information (generated continuously in a stream) and pulling old one seems winning: push operations open the receiver window time frame (or chunk trading window) with new chunks, and pull operations try to fill holes left by non contiguous pushes.

We remark that nodes have no information on target nodes' status and signaling is limited accepting or refusing a transfer. To explore the importance of choices, we also introduce a small window choice in chunks offers and requests: peers propose ω chunks that can be pushed or that wish to pull.

3.6 Performance Metrics

Initial investigation is focused on:

Maximum diffusion delay We recall that the maximum propagation delay δ_{max} is the delay to diffuse any chunk in the entire overlay.

90-th percentile of the transfer delay The 90-th percentile of the transfer delay δ_{90}. is the delay after every nodes has received the 90% of chunks, indicated with

Parameter	Value
$\lvert\mathcal{K}\rvert$	10^3
$\lvert\mathcal{C}\rvert$	10^4
$\mathcal{C}_{bits}$	122 kbit
B_{UP}	192, 220, 256 kbit/s
$\lvert\mathcal{N}\rvert$	16
$\omega_{push} = \omega_{pull}$	1, 2, 4
ρ_{dw}	1, 4

Table 3.2: Simulation parameters values.

Histogram of the diffusion delay This histogram represents an approximation of the measured probability density function (pdf) of the chunk diffusion delay.

3.7 Experimental Results

We have implemented our protocol on PeerSim adding the priority-sharing bandwidth management mechanism. We consider several parameters: network size, number of chunks, upload and download bandwidth, minimum and maximum one-way delay, number of uploads and downloads and finally the maximum number of chunks offered in either push or pull. The chunks size $\mathcal{C}_{bits}$ is set equal to 122 kbit and we remark that source has the same resources of other peers. The simulation parameters are reported in Table 3.2.

Fig. 3.8 shows a comparison between δ_{max} and δ_{90} varying the upload bandwidth without parallel download. The results show that with upload bandwidth $B_{UP} = 1.5$ times the stream rate B_s, δ_{max} is less than 20 s and obviously δ_{90} is lower than the last one, which are values allowing VCR-like streaming. As we expect, by increasing the upload bandwidth both values decrease. We recall that this study is exploring a basic version of this protocol

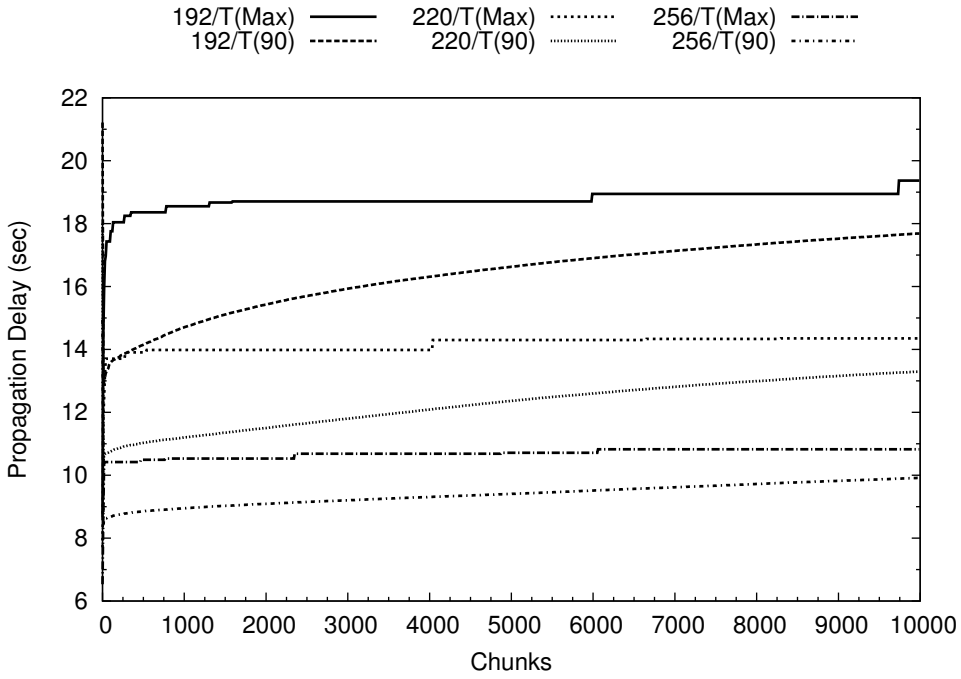

Figure 3.8: δ_{max} and δ_{90} for different uploads: $|\mathcal{K}| = 1000$, $|\mathcal{N}| = 16$.

which retrieves all chunks, we are currently exploring solutions. Both δ_{max} and δ_{90} can be used for tuning buffering for different quality desired, in particular δ_{90} is useful for dimensioning MDC or FEC codes.

Finally, Fig. 3.9 reports the histogram of chunk diffusion delay varying the chunks window ω. We observe that parallel downloads and choice on chunks selection influence the system performance, reducing the chunk diffusion delay.

3.8 Lesson Learned

This basic version of a Push/Pull protocol is coupled, intentionally, with random selection of peers. We observe that enabling chunks choice during negotiation and parallel downloads, the protocol reduces the chunk diffusion delay, revealing interesting potential to support streaming in P2P systems.

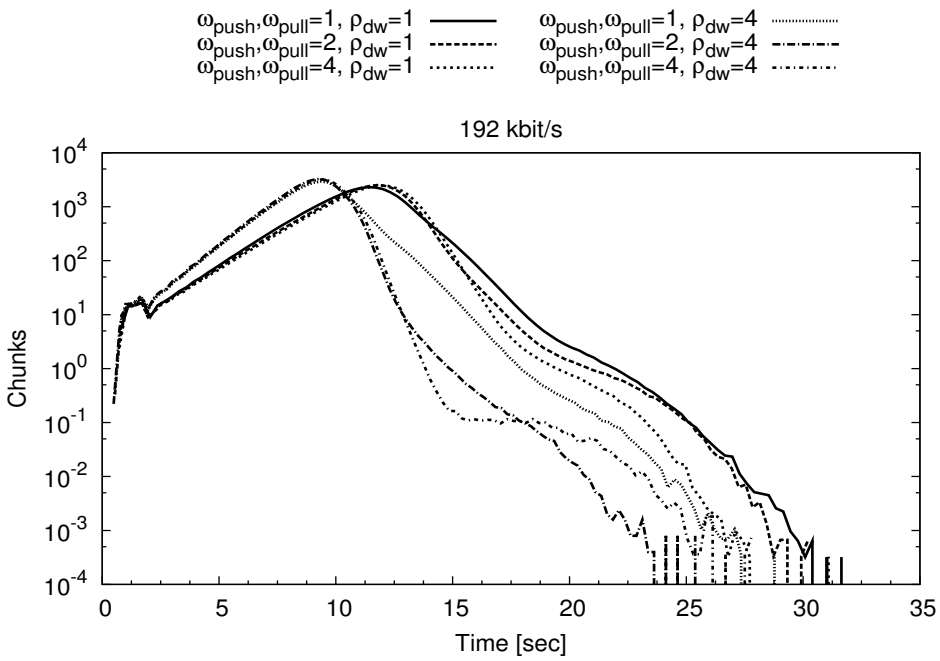

Figure 3.9: Chunk diffusion delay histogram: $|\mathcal{K}| = 1000$ and $B_{UP} = 192\,\text{kbit/s}$.

3.9 The Importance of Network-Aware Protocols

We recall that each peer $\mathcal{K}_i$ is completely independent from the others in its decisions and there is no timing coordination, apart from the fact that chunks are timestamped from the source. We remark that this definition differs radically from the asymptotic analysis of [85], where global coordination is considered and the system works in *cycles* with all peers being synchronized either in Pull of Push state. We usually refer to state with capital letter (e.g., Push) while the messages are with small letters (e.g., pull).[3]

In general, Push/Pull protocols work as described in Algorithm 4 and peers can follow any algorithm to select the chunk and peer for scheduling either the push or the pull (functions ChunkSelect and PeerSelect in Algorithm 4).

[3]This work is supported by the European Commission through the NAPA-WINE Project (www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant No. 214412.

Part of this work was published in the proceedings of the IEEE International Conference on Communications (ICC 2010), Cape Town, South Africa [5].

Algorithm 4 Generic algorithm for Push/Pull protocols at node $\mathcal{K}_i$,

```
1: at every peer K_i
2: repeat
3:     if State == PUSH then
4:         K_j = PeerSelect(N_i)
5:         C_c = ChunkSelect(C(i))
6:         Transmit(K_j,C_c)
7:         State = NextState(PUSH,Transmit)
8:     else if State == PULL then
9:         K_j = PeerSelect(N_i)
10:        C_c = ChunkSelect(C(i))
11:        Receive(K_j,C_c)
12:        State = NextState(PULL,Receive)
13:    end if
14: until the stream is finished
```

Each peer verifies if the exchange is useful with a signaling message before proceeding to the actual chunk transfer itself. Finally, the protocol will decide the next Push/Pull state as a function of the current state and the result of the previous exchange.

Fig. 3.10 exemplify the protocol data exchange for a successful push and pull operation, unsuccessful operations are Ack-ed explicitly.

We note in this simple version the alternation of push and pull, this mean that each node may be either in push or pull state, while the protocol may also execute both push and pull in parallel, in according to node's state.

We are interested in protocols where peers do not exchange information about their state prior to scheduling so that the peers do not maintain any status relative to their neighbors, making the protocol inherently robust to dynamic neighborhood management. Specializing the Algorithm 4 to operate without knowledge about neighbors' state and with the alternation logic just described, the Algorithm 5 is obtained: each node actively contributes to the streaming by propagating fresh information via Push (*Latest* function); the

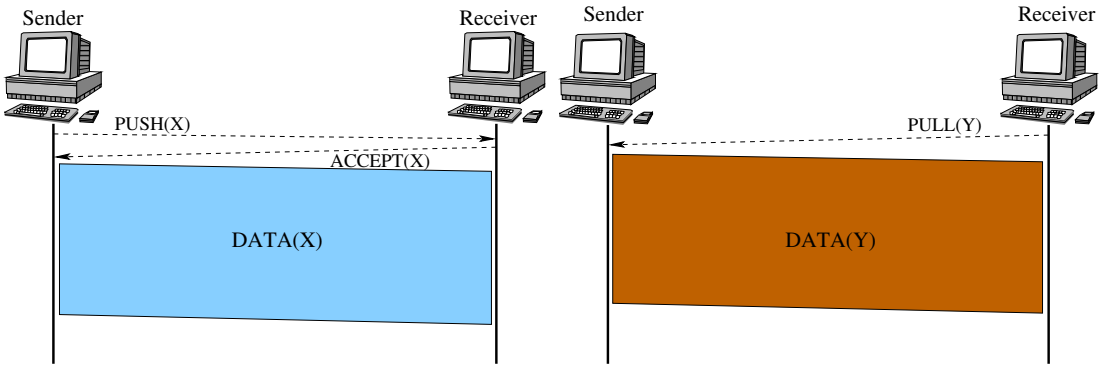

Figure 3.10: Example of push (left) and pull (right) communications.

Pull is devoted to retrieve locally missing chunks (*OldestMissing* function), thus it does not contribute to the streaming, but only to enhance local quality.

Algorithm 5 defines the *active* behavior of peers, when they propose an exchange. In particular, during Push, each node offers a set of chunks to some neighbors, actually pushing only to the first neighbor that accepts, while the others are refused explicitly; in Pull, the node requests a set of chunks to its neighbors, accepting only the first positive reply, refusing the others. The protocol description is completed by the *passive* node's behavior, when a node is queried by other peers: Each node may receive many offers of push, accepting only missed chunks and refusing the others, while it satisfies only one pull requests among all received.

Fig. 3.11 shows a graphical representation of node's state. Obviously, a node can chose its own active state, so that it can decide to be either in Push or Pull state, but it cannot control the type of queries it receives, so that alternating between Push and Pull when queried it is not possible.

A window of ω chunks are offered or requested by peers during negotiation, giving some choice to the queried node, hopefully increasing the efficiency of the protocol. The node's state is changed deterministically from Push to Pull and vice versa upon success, or if the number of failed communication attempts is equal to n_a. Notice that a node can signal its offers and requests in

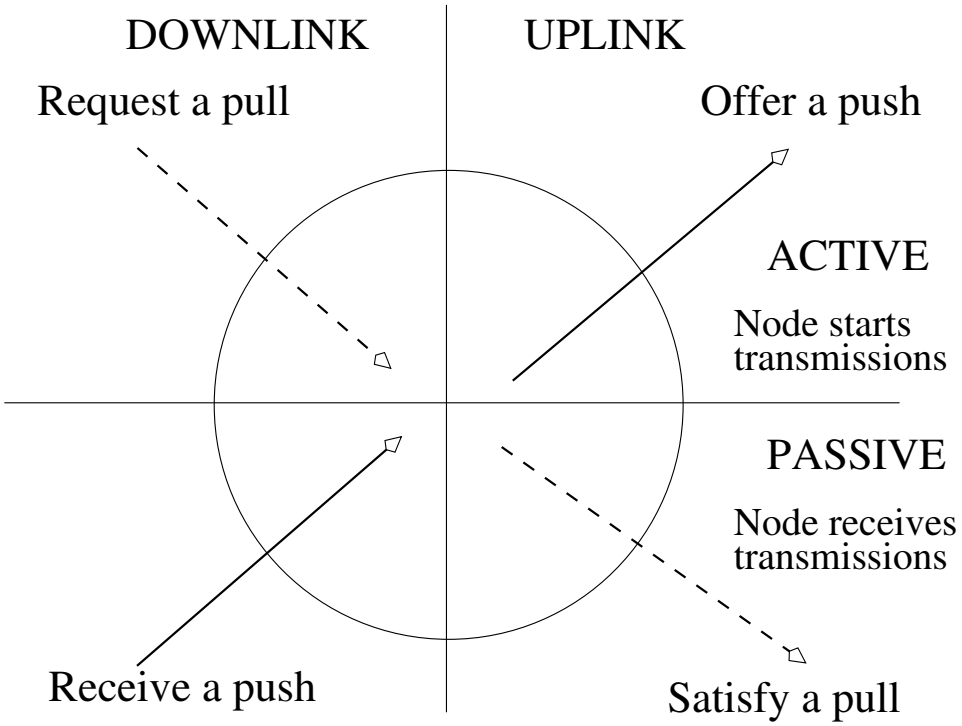

Figure 3.11: Active and passive states for a peer in Push/Pull protocols

parallel to multiple nodes to enhance its success probability. The higher the parallelism the higher the signaling overhead, but the higher the probability of success too. Considering that a signaling message is normally a single small packet, while a chunk can contain several hundreds kB of data, the signaling overhead remains small.

Albeit it is possible to execute Push and Pull phases in parallel (i.e., each node may send push and pull messages at the same time), we decided to maintain the alternation between the two states to limit the complexity of the system. This means that each node may be either in active Push or in Pull state, while the passive state depend on other nodes requests. We remark that the rationale behind this stateless behavior is simple, yet compelling: The push operation disseminate recent information generated by the source, while pulling tries to fill the gaps in owned information.

Algorithm 5 Revised Push/Pull protocol with peer/chunks selection.

```
1: at every peer K_i
2: repeat
3:     K_j = UniformRand(N_i)
4:     if State == PUSH then
5:         {C_c} = Latest(C(i),ω)
6:         Transmit(K_j,{C_c})
7:         if Tx-success OR Tx-failures==n_a then
8:             State = PULL
9:         end if
10:    else if State == PULL then
11:        {C_c} = OldestMissing(C(i),ω)
12:        Receive(K_j,{C_c})
13:        if Rx-success OR Rx-failures==n_a then
14:            State = PUSH
15:        end if
16:    end if
17: until the stream is finished
```

As a consequence, we identify four states in a node, based on its active/-passive state, as exemplified in Fig. 3.11: push/push, push/pull, pull/push, and pull/pull. We observe that the actual active state is only one, since node alternates push and pull, while the passive state is independent. This means that the node may perform a push (actively) while it is receiving a chunk via push and (potentially) is satisfying a pull, sharing its upload bandwidth.

A peer has the freedom to accept or refuse queries. We indicate with ρ_{up} the maximum number upload connections (exchanges that insist on the uplink of the node) that a node can accept to satisfy Pull requests, and ρ_{dw} the maximum number of *passive download* (exchanges that insist on the downlink of the node) that can be activated when a peer receives Push offers. The value of ρ_{up} and ρ_{dw} may depend of up-link and down-link resources or simply on peer's policies.

3.10 Delay-Aware Peer Selection

Algorithm 5 selects one peer in the neighborhood at random. However, this choice neglects network conditions, making the P2P distribution system network-agnostic: Performance suffers and the network is often overloaded. As observed in the NAPA-WINE project [18], there are many network properties and characteristics that can be exploited to improve the system behavior, however, most of them are very difficult to measure, such as the available bandwidth between two peers. The round-trip-time (RTT) delay instead is very simple to measure accurately and is normally a very good indicator of how far nodes are and how much congested the network is between them. The RTT is the time required for a packet to travel from a source node to a destination node, back again. Thus, trying to communicate with close-by (in terms of RTT) peers should improve performance and have a lighter impact on the network.

The main difference between the random and the delay oriented peer selection is that the second one picks those peers with lower RTT with higher probability, promoting exchanges with closest neighbors. A simple way to do this is selecting peers in the neighborhood $\mathcal{N}_i$ based on a probability distribution function build as follows:

$$V_{i,j} = \frac{1}{RTT_{(i,j)}}, \quad i,j \in \mathcal{K} \tag{3.1}$$

$$SV_i = \sum_j V_{i,j}, \quad j \in \mathcal{N}_i \tag{3.2}$$

$$P_i(j) = V_{i,j}/SV_i \tag{3.3}$$

This choice reduces the average delay $\overline{\delta}_{peers}$ required to exchange each chunk between any peer, which in turn will significantly reduce the overall diffusion delay, since the diffusion delay is comprised between $\overline{\delta}_{peers}|\mathcal{K}|$ for systems based on a distribution chain, and $\overline{\delta}_{peers}\lceil \log_2 |\mathcal{K}| \rceil$ for optimal system that double the number of chunk replicas at every new transmission [12].

In some cases, the delay oriented peer selection may suffer from repeated selection of the same neighbors, that reduces the fairness, and thus, the correct spreading of information, building chains of peers instead of swarms. For this reason, the peer selection is poisoned, so that the same peer can be selected again only after $2\overrightarrow{T_S}$. We assume that a chunk can be transmitted within $\overrightarrow{T_S}$, and we recall that nodes alternate Push and Pull, so the maximum time between two Push phases in the same node is $2\overrightarrow{T_S}$, while on average multiple Push and Pull will alternate in this time. Without this threshold, a peer may push two or more consecutive chunks to the same peer, which may end up propagating only the last one via Push, while the others linger behind and will finally be retrieved via Pull, increasing their diffusion delay as a consequence. This poisoning guarantees a correct distribution of fresh content via Push.

3.11 Performance Metrics

The following performance metrics are investigated:

CDF of the average chunk diffusion delay The average diffusion delay for chunk c among all peers is

$$\overline{\delta}_c = \frac{\sum_p \delta_p(c)}{|\mathcal{K}|}, \quad c \in \mathcal{C}, p \in \mathcal{K} \tag{3.4}$$

Then, we compute the Cumulative Distribution Function (CDF) over all chunks.

CDF of the average transfer delay among peers The average chunk diffusion delay at peer p is

$$\overline{\delta}_p = \frac{\sum_c \delta_p(c)}{|\mathcal{C}|}, \quad p \in \mathcal{K}, c \in \mathcal{C} \tag{3.5}$$

We then compute the CDF over all peers.

Histogram of the chunk diffusion delay It is the measured probability density function (pdf) of chunk diffusion delay averaged over realizations. Here, each chunk, at each peer, provides a point of the estimated pdf. Thus, each realization provides $|\mathcal{C}| \times |\mathcal{K}|$ points. The bin size unit is 0.5s.

3.12 Experimental Setup

We have implemented the Push/Pull protocols described in Section 3.9 in PeerSim [74].

The simulator structure is based on modular programming which allows adding new components, facilitating protocols implementation, moreover it easily supports a large number of nodes. The Push/Pull protocol extension as well as the network model we implemented, and all other "side code", are available to the community and will be shortly added to the PeerSim distribution.

The overlay topology graph $G(\mathcal{K}, \mathcal{E})$ is randomly built, in general having an n-regular random graph as a good model. Indeed, the topologies considered in different works are not always n-regular topologies; however, most of the properties considered for mesh systems are not strictly dependent on the specific topology, whose main property is the randomness with a roughly constant connectivity degree, as confronted to specific, deterministic topologies such as trees or hypercubes (example in Fig. 3.12). The topology is built selecting $|\mathcal{N}| \in \mathcal{K}$ neighbors at random for each node, with symmetrical connections: $\mathcal{E}(i,j) \rightarrow \mathcal{E}(j,i), i,j \in \mathcal{K}$ ($\mathcal{E}(\cdot,\cdot) \in \mathcal{E}$ by construction). We use $|\mathcal{N}| = 16$; larger neighborhoods should improve performance.

As parameter to tune the application to the network we consider the RTT for several reasons: i) The signaling phase introduced before chunks exchange, to avoid conflicts and wasted transmissions, is heavily affected by delay; ii) RTTs between peers are easily measured even at the application layer and

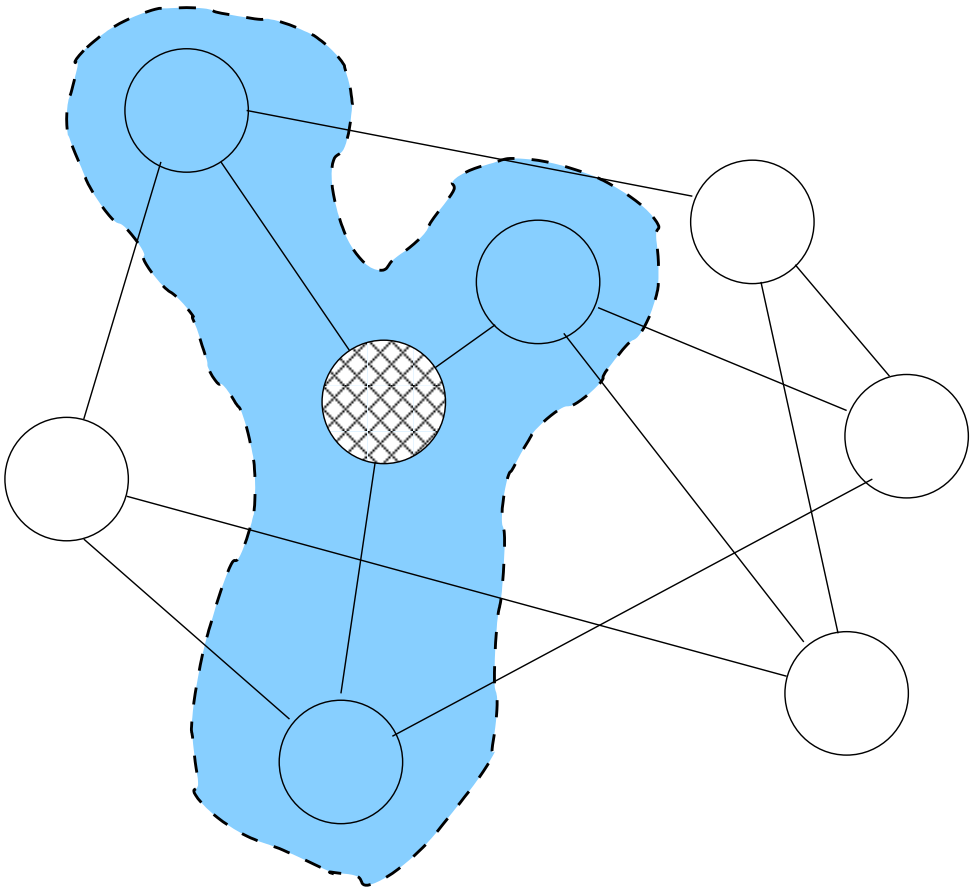

Figure 3.12: A 3-regular topology with eight nodes, the shaded area is the neighborhood of the black node.

are not much affected by correlations; iii) RTT delay is an indication of how far peers are in terms of hops, so that choosing closer peers will also reduce the global amount of network resources used.

We assume that the size of control messages is negligible (a few tens of bytes), so that the message transfer delay on every connection $\mathcal{E}(i, j)$ is roughly 1/2 of the RTT. In the simulations we model the RTT of connections with a random variable extracted from a uniform distribution between $[\delta_{min} \div \delta_{max}]$ ms. RTTs from peer i to peer j and vice versa are the same.

We indicate the maximum number of parallel signaling messages sent by the node in active mode with α: α_{up} indicates messages that involve up-

loading (push offers), while α_{dw} is for messages involving downloading (pull requests). The passive node's behavior is controlled by ρ, which indicates how many of the messages received are positively answered: ρ_{up} refers to accepting requests that use the upload bandwidth (satisfying a pull), while ρ_{dw} concerns the download usage (receiving pushes).

The source $\mathcal{S}$ streams at B_s Mbit/s, generating a new chunk every $\overrightarrow{T_{\mathcal{S}}}$ s which is sent to the other peers. We stress that the source is a normal peer without additional resources; This is important because many evaluation studies assume that the source is somewhat special and has more upload bandwidth, which indeed modifies the performances non marginally. In order to guarantee the continuous emission of chunks, both negotiation and transmission of each chunk should be done within $\overrightarrow{T_{\mathcal{S}}}$s. In accordance to these considerations, the peers upload bandwidth is always $B_{UP} \geq \overrightarrow{T_{\mathcal{S}}}/(\overrightarrow{T_{\mathcal{S}}} - \delta_{max})$, in which is normalized to B_s. In our simulation study we use $\overrightarrow{T_{\mathcal{S}}} = 1$ s, thus for $\delta_{max} = 500$ the minimum value of B_{UP} is 2.

Bandwidth availability and its sharing in the up- and down-link are critical parameters for streaming applications. In actual networks the sharing is mediated by the transport protocol (TCP, UDP, UDP/RTP, etc.); However, PeerSim does not support actual packet transmission (which is one of the reason it allows simulating large overlays). We implemented a bandwidth management system based on priority sharing, which gives as much resources as possible to the first connection, then it tries to satisfy the second with the remaining part, and so on. This bandwidth mechanism is accurate enough to capture chunk transfer interaction and simple enough to maintain high simulation speed.

We consider a mildly popular stream with $|\mathcal{K}| = 1000$ and $|\mathcal{C}| = 4000$. We limit the accepted number of downloads to ten ($\rho_{dw} = 10$), allowing parallel downloads, and each node can satisfy at most one pull per time ($\rho_{up} = 1$). In Push each node proposes a window of size ω_{push} chunks to its neighbors,

Parameter	Value
$\|\mathcal{K}\|$	10^3
$\|\mathcal{C}\|$	4×10^4
$\|\mathcal{N}\|$	16
$\overrightarrow{T_S}$	1 s
ρ_{up}	1
ρ_{dw}	10
α_{dw}	1, 4
α_{up}	1
$\omega_{push} = \omega_{pull}$	1, 4
$[\delta_{min} \div \delta_{max}]$	$[10 \div 250]$, $[10 \div 500]$ ms

Table 3.3: Parameters used for evaluating delay-awareness.

while in Pull it requests a set of ω_{pull} chunks. We usually set $\omega_{push} = \omega_{pull}$, referring to them with ω, as well as $\alpha_{up} = \alpha_{dw}$, indicating their values with α. In this study, we limit the maximum number of attempts $n_a = 1$.

We investigate bandwidth homogeneous networks to simplify results interpretation. The system is at steady-state (peers in the overlay are stable). Peers try to retrieve all chunks of the stream, so that we can evaluate the tail behavior of the system; if an application may tolerate, say, 5% losses, it is enough to read pdf plots at 95^{th} percentile, as we do in Fig. 3.15.

3.13 Results

We discuss a set of results highlighting the role of the Push and Pull phases in the information delivery, and results showing the impact of RTT on the delay experienced by peers through the comparison of the Random (R) and Delay based (D) peer selection. Further details on this study can be found in [6].

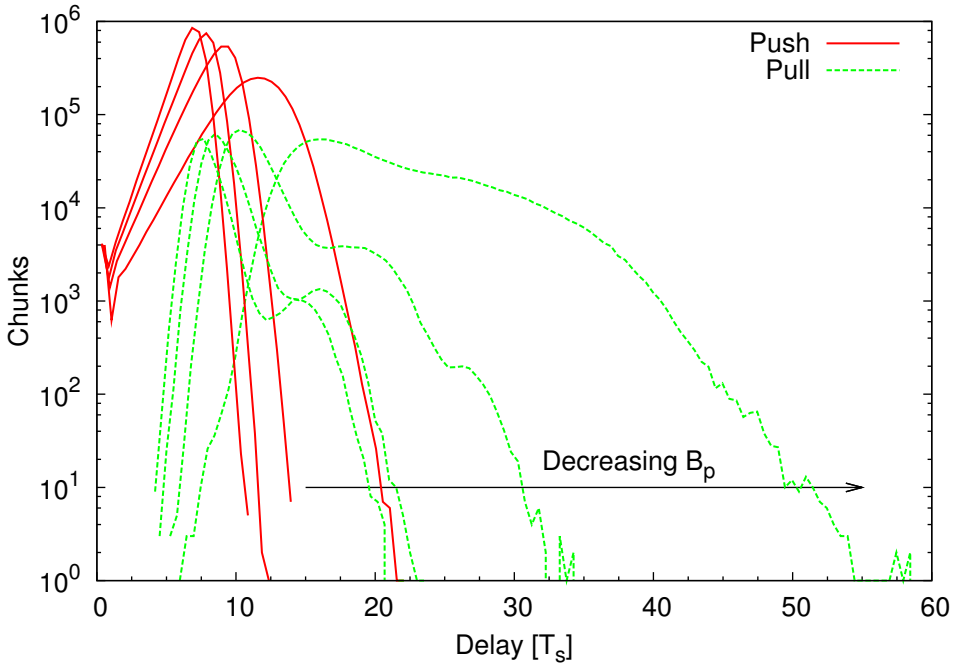

Figure 3.13: Histogram of chunk diffusion delay for Push and Pull mechanism, for $B_{UP} = \{1.9, 2.4, 2.9, 3.4\}$, and RTT $[10 \div 250]$ms.

3.13.1 The Role of Push and Pull

Fig. 3.13 highlights the different role of the Push and Pull states, and how they contribute to the pdf of chunks distribution. The push mechanism starts the diffusion before the pull, limiting the maximum delay and it ends before, while the pull starts to retrieve data when there are more chunks in the network, so the pull requests could be satisfied. We can see that Push phase has lower delay and spreads the greater part of the chunks, while the Pull operates filling the gaps of missed chunks, so that chunks retrieved via pull are those experiencing the largest delay, while the Pull experiences the largest delay due to blind search for retrieving the missing chunks. We expect such a behavior because the Push is the active part of the distribution process, while Pull is a local recovery mechanism. The quantitative difference is however much larger than one can expect. The shape of the pulled chunks delay distribution shows lighter tails as B_{UP} increases, since many more chunks are

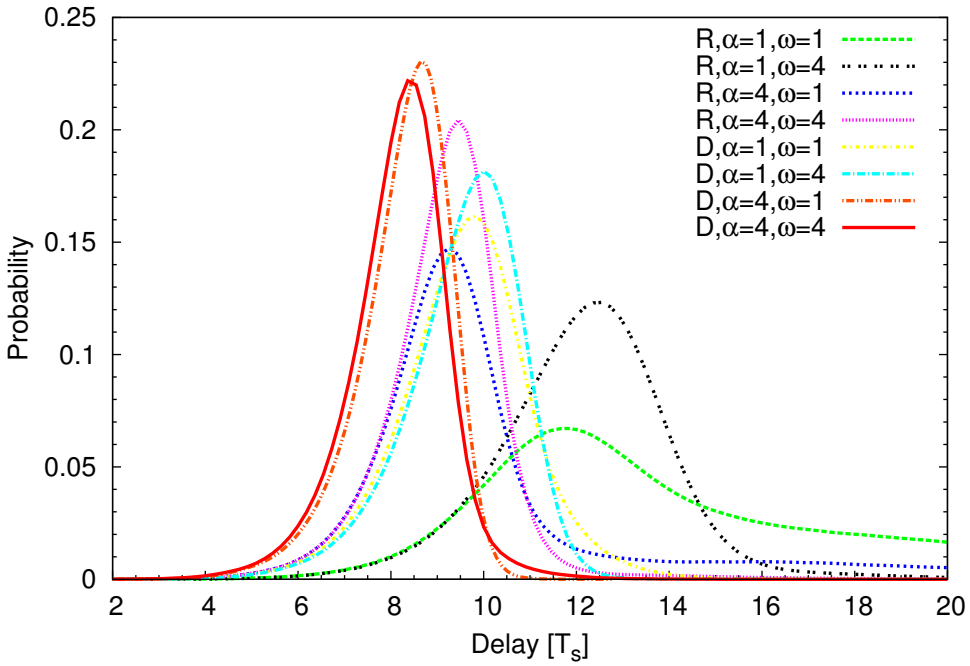

Figure 3.14: Histogram of chunk diffusion delay among all peers varying ω, α: B_{UP}=1.9, and RTT $[10 \div 250]$ ms.

diffused via Push, so the number of pulls become smaller and the probability to find a chunk via pull becomes higher.

3.13.2 Impact of Choice and Parallel Signaling

Fig. 3.14 shows the effect of varying the number of messages issued in each phase, and of providing a window of ω possible chunks during negotiation. First we focus on random peer selection, and then we describe the delay oriented one. We fixed $B_{UP} = 1.9$, because in this situation the basic Push/Pull protocol ($\alpha = 1$; $\omega = 1$) does not provide satisfactory performance, as shown by the dashed line in Fig. 3.14. For $\alpha = 1$ and $\omega = 4$, offering multiple chunks leads to a clear reduction in diffusion delay, and the tail becomes smaller, as shown by the two-dotted curve. When we set $\alpha = 4$ and $\omega = 1$, the protocol achieves lower average diffusion delay, but the distribution tail remain important (short dashed line in Fig. 3.14), because some chunks are not properly

diffused and remain rare. The combination of parallel signaling ($\alpha = 4$) and more choice among chunks negotiation ($\omega = 4$), reaches satisfactory results (dotted curve). The corresponding curve has a negligible tail above 12 s, and its mass is well concentrated around the mode at 9.55 s.

Now, we focus on the delay aware peer selection, always achieves better results, with negligible tails. In particular, with $\alpha = 1$ and $\omega = 1$ the delay aware selection limits the diffusion delay to roughly 14 s, while the corresponding curve with random selection has a tail extending beyond 20 s. When $\alpha = 1$ and $\omega = 4$, the curve becomes narrower and within 12 s, while for $\alpha = 4$ and $\omega = 1$, the curve shifts to the left, obtaining a meaningful diffusion delay reduction. Finally, for $\alpha = 4$ and $\omega = 4$, the combination of parallel signaling and the possibility of choosing among chunks, with the delay oriented peer selection, leads to lower delays than all the others, as shown by the solid line. This plot confirms several important points. Although no information is exchanged during negotiation and without multiple messages, the Push/Pull protocol is able to limit the delay experienced by peers, also when the random peer selection is used. The ability to choice during chunks negotiation reduces the chunks' delay not marginally, in particular, for those tails due to chunks not well distributed: this is particularly clear for random peer selection from $\omega = 1$ to 4.

3.13.3 Impact of Upload Bandwidth

We continue our analysis allowing both parallel signaling and allowing choice during chunks negotiation, i.e., with $\alpha = 4$ and $\omega = 4$. Fig. 3.15 presents the 95^{th}-percentile of chunk diffusion delay, for different uploads configurations, and RTT distribution $[10 \div 250]$ and $[10 \div 500]$ ms.

Again, the delay aware peer selection shows that our intuition is right, achieving an average diffusion delay lower than the random peer selection one, especially with small upload bandwidth. Increasing the maximum RTT,

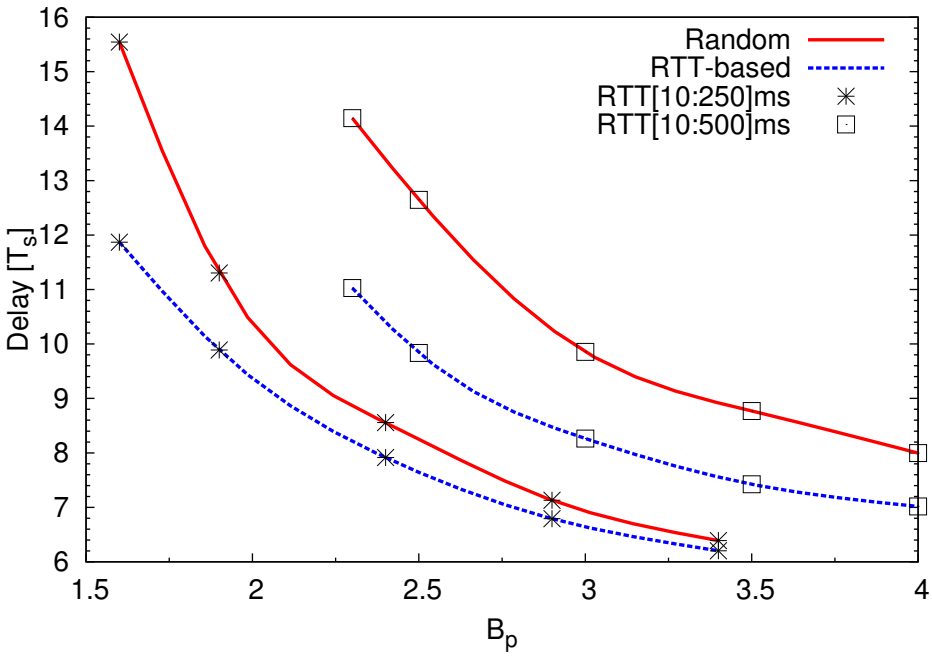

Figure 3.15: 95^{th}-percentile chunk diffusion delay for different B_{UP} and RTT.

the delay aware selection shows a larger gain even when bandwidth resources are very high. Although we double the RTT delay, the delay aware selection does not reflect a proportional increase in chunk diffusion delay for the same upload bandwidth: the difference remains around 1.5 times.

In Fig. 3.16 we focus our attention on RTT between $[10 \div 500]$ ms, showing its impact for different upload bandwidth. In this case, the RTT heavily impacts chunk diffusion delay, and increasing the upload bandwidth the RTT becomes the main part of chunk diffusion delay. For this reason neighbors selection should be done carefully, reducing the impact of the RTT and its propagation in chunks' distribution. The curves become narrower as B_{UP} increases, for both random and delay oriented selection. However, the delay oriented approach always achieves lower delay than the random one, reducing the average chunk diffusion delay experienced by peers, giving useful information for computing buffering time and timeout for chunks to be pulled.

Fig. 3.17 shows the comparison between the delay oriented and the random

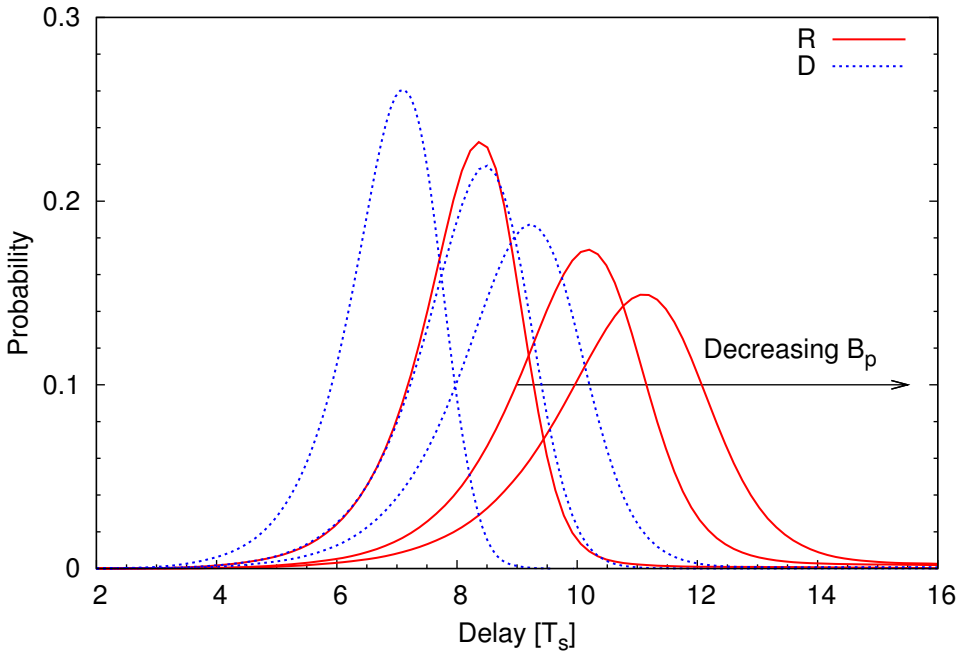

Figure 3.16: Histogram of chunk diffusion delay for RTT $[10 \div 500]$ ms and $B_{UP} = \{2.3, 2.5, 3.0\}$.

peer selection, comparing the cdf of the average delay experienced by peers during the streaming. The delay experienced by peers are quite stable, and the curves rise quickly, showing that the inter-arrival chunks delay in each peer is small and steady. Thus, the delay oriented approach confirms that the choosing of closest neighbors leads to low delay, without big dispersion. Note that with just $2.3B_s$, with RTT $[10 \div 500]$ ms, the delay oriented is within 12 s, while the corresponding random is around 16 s, and the average distance between them is about 30%.

3.14 Conclusion

In this chapter, we discussed some fundamental properties of hybrid Push/Pull protocols for P2P streaming applications with zero state, i.e., nodes have only limited local knowledge of their neighbors. Zero state means

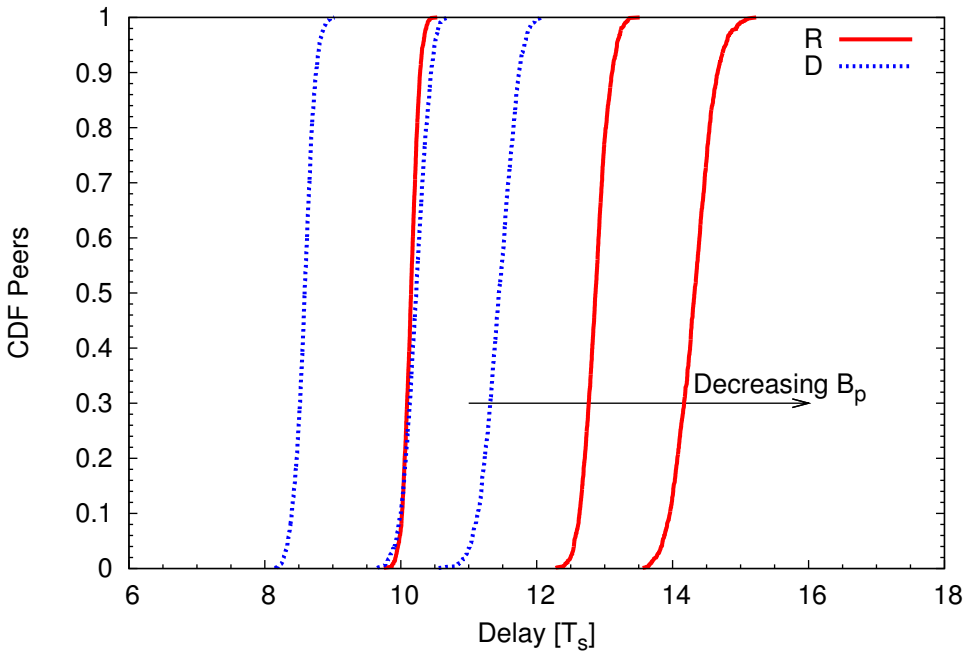

Figure 3.17: CDF of the $\overline{\delta}(p)$, $B_{UP} = \{2.3, 2.5, 3.0\}$, and RTT $[10 \div 500]$ ms.

that the algorithms used to select chunks and peers rely solely on local information and the state of neighboring nodes is not known to the node taking the decision. The core of the protocol is the alternation of Push and Pull phases, where the chunk to be pushed is the most recent owned by the peer, while the chunks to be pulled is the oldest not yet received by the peer.

The protocol is able to work without making any assumption on the node behavior, because each transmission occurs after the corresponding negotiation, making the protocols suitable to situations with network dynamics. We evaluate the performance of this basic scheme, under symmetric and asymmetric topologies, exploring the impact of the upload bandwidth and the network size, moving from the cycle-based model to a more realistic asynchronous model.

In addition, we explored how adding some choice to the system, i.e., the possibility for the selected peer to choose among a small set of chunks to be pushed or pulled, as well as issuing more push or pull messages in parallel,

influences the overall performance.

Finally, a key result of delay-awareness is that RTT is a very important parameter for delay sensitive systems, so promoting communication among closest neighbors are more suited, even with limited resources. We stress that the proposed system is bandwidth-optimal in that the only useful information (and signaling) is distributed, avoiding the waste of many P2PTV systems. Moreover, the ability to choose during chunks negotiation, combined with parallel signaling, lead to lower diffusion delay.

The overall insight gained on the Push/Pull system shows that with resources which are no larger than two times the stream rate, the streaming is sustainable and efficient (the protocol ensures small overhead), a result that with protocols exploiting only push or pull mechanism is achieved only at the price of state exchange between peers.

Further research include exploring churn impact, heterogeneity of nodes, different overlay construction mechanisms, Peer and Piece scheduling algorithms, and more efficient implementations of the Push/Pull protocol, e.g., by allowing signaling to be in parallel with data transmission.

Chapter 4

Cooperative Wireless Mesh Multicasting

4.1 Introduction

Multimedia communications to mobile devices are one of the most powerful drivers of the networking and communications market. Be it standard on-demand streaming, live video events or on-line gaming, the availability of good quality video on tablets, smartphones and mobile devices in general is one of the key issues in the selection of the device to buy and the network to subscribe to. Most multimedia communications are now integrated over the Internet, but when it comes to broadcast or multicast services the global architecture of the Internet remains badly adapted, and a wireless last-leg (be it 3G, LTE or WiFi) to ensure mobility does not help.

The use of multi-hop, or mesh, wireless networks is rapidly expanding due to their low cost and high bandwidth availability, their intrinsic support for proximity and location based services, and their simple deployment, often not requiring licensing permits or high capital investments. Wireless Mesh Networks (WMNs) are ideal for outdoor deployment and cheap, compared to wireline networks, because are self-configuring and self-healing, limiting the cost of administration and maintenance. Thus, they are perfect for public

safety, municipalities, and industrial purposes. They can be used to deploy remote monitoring and emergency communications services, as well as smart cities, and providing network coverage those users that are difficult to connect because of their geography. In WMNs each node has a limited view of the whole system, communicating with just several of its neighboring nodes wirelessly. Hence, routing mechanisms must be used to direct packet flows. Multicast distributions cannot simply rely on the broadcast nature of a single shared wireless medium.

Several protocols have been proposed to address multicast routing in ad-hoc networks [53, 91, 90]. In the latter papers, the authors have constructed efficient multicast mechanisms for mobile ad-hoc wireless networks through the dynamic synthesis of mobile backbone networks. In [26], authors observe how the validity of multicast routing protocols designed for wireline networks is undermined by the broadcast nature of the wireless channel and network dynamics. In [96], the authors consider the use (for wireless multicast) of modified versions of commonly employed wireline multicast routing protocols such as the Distance Vector Multicast Routing Protocol (DVMRP), Multicast extension to Open Shortest Path First (MOSPF), and Protocol Independent Multicast (PIM).

Multicast participants subscribe a specific multicast session they are interested in. A session is formed to include the users that are involved in interactive communications relating to certain applications. A source user that has joined the session is then able to communicate with clients that are members of the same multicast session. The multicast routing protocol is in charge of distributing packets across the network to other members of the multicast session. Routers that are elected members of this multicast session are configured to forward packets to other routers in the same session. Multicast group end-users are attached to some of the routers that are included in the multicast session. Packets of the multicast session are normally flooded on the distribution tree that includes all the routers in the session.

Extending standard wireline Internet multicast protocols to mesh wireless networks is not straightforward, due to the intrinsic differences between the underlying media. Several studies [60, 61, 104] have pointed out that the execution of multicasting in a wireless network under the use of a PIM protocol does not lead to acceptable performance behavior. Nevertheless, to the authors' knowledge, only few works have engaged in studying the use of the PIM-SM protocol [54, 107], across wireless networks; and we know of no study that presents analysis of the PIM-DM scheme over wireless ad-hoc networks. Also, its operation over mesh wireless networks has not yet been addressed.

The PIM protocol [31] defines a class of multicast routing protocols which are independent of the unicast routing protocol that is employed. PIM protocols impose the construction of a multicast tree by using the underlying unicast routing tables, assuring the coverage of all distributed session participants. Since building a multicast tree has distinctly different flavors based on the density of the network's destination hosts, two PIM protocol versions have been identified: Sparse Mode (SM) and Dense Mode (DM).

In the first part of this chapter, we discuss whether the PIM-DM protocol scheme can be used over wireless ad-hoc networks as it currently stands, and identify the limitations of its operation we discuss and evaluate the use of the PIM-DM protocol scheme to support multicast applications within a single domain, (e.g., campus or park), using a wireless network. Over such a domain, it is assumed that the designated multicast stream is directed to a large number of receiving nodes that are densely located.

The contributions provided in this work include the following: *(i)* We discuss whether the PIM-DM protocol scheme can be used over wireless ad-hoc networks as it currently stands, and identify the limitations of its operation; *(ii)* We present a simple mechanism for resolving the identified limitations, enabling the PIM-DM based scheme to properly operate over wireless mesh

networks. We perform this task by introducing simple modifications to the protocol's management process, while not changing the protocol itself, so that it remains compatible with the underlying Standard recommendation; *(iii)* We show how the characteristic features of PIM, such as its protocol independence and easy implementation properties, can be effectively exploited; and *(iv)* We present results that depict the performance of the protocol when used over regular wireless mesh networks, providing insight into the process to be used in planning and designing such networks.

The reason for focusing on the PIM-DM protocol is explained by the following. Considering the networking of packet flows over a wireless mesh networks that are generated by using applications such as live streaming events and gaming oriented processes, we note that user locations will be highly correlated. Consequently, many user nodes that are concentrated over distinct neighborhood clusters will tend to share the multicast tree. Consequently, a dense mode operation is applicable. In turn, over the global Internet, the use of the PIM-SM protocol is employed when conditions are such that the density of user nodes that are interested in a specific multicast, compared to that of the routers, is rather small, resulting in a sparse network.

The rest of this chapter is organized as follows. Section 4.2 provides a brief description of the PIM-DM protocol, identifying mechanisms that adapt its operations to a wireless network. Sections 4.3 and 4.4, describe the evaluation methodology and the experimental setup, respectively. Section 4.5 presents the results that confirm the performance efficiency over wireless networks of the Wireless-PIM-DM protocol, as modified by the use of the proposed adaptations. Section 4.6 presents several comments on the Wireless-PIM-DM protocol. Section 4.7 describes cooperative protocols for wireless mesh networks. Section 4.8 reviews the related works. Section 4.9 provides helpful observations for promoting cooperation among end-users in WMNs. Section 4.10 introduces PullCast, a peer-assisted protocol for wireless mesh

multicasting. Section 4.11 provides the performance metrics, and Section 4.12 describes the simulation model and the scenarios investigated. Section 4.13 evaluates PullCast performance and feasibility. Finally, Section 4.14 summarized the conclusions of this chapter, providing future work directions.

4.2 Enabling PIM-DM over WMNs

PIM-DM is a data driven protocol that is designed to support multicast sessions where the end-nodes are densely distributed in the network; i.e., the probability that a multicast router has end-nodes attached to it is high. The approach used is based on performing flooding and pruning operations, rather than defining rendezvous points: the source node floods the network with multicast packets and the routers apply the forwarding rule. Whenever a leaf router has no end-user clients for a specific multicast group (or session), it sends a PIM Prune message towards the upstream router; this is used to prune the link branch connecting this leaf node across the multicast tree to this upstream neighbor node. PIM-enabled routers that execute the PIM-DM protocol are called PIM-routers; they construct the multicast tree from the source node towards all PIM-routers that have associated end-nodes, called PIM-clients. PIM-clients join the multicast group, so that the nodes involved in a multicast session are either PIM-routers or PIM-clients.[1]

PIM-DM has been designed for wired networks and it is based on the use of routers' interfaces. These are classified into two categories: upstream and downstream. The upstream interface is the one connecting the router to the next hop router toward the source node, based on the use of the reverse path forwarding (RPF) process. The others are identified as downstream

[1]Part of this work was published in the proceedings of the IEEE International Conference on Computing, Networking and Communications (ICNC 2013), San Diego, CA, USA [9].

This work was partially done while the author was a visiting Ph.D. student at the University of California, Los Angeles (UCLA).

interfaces. The forwarding rule is defined in terms of interfaces: whenever a data packet is received from and upstream interface $\mathcal{U}_{\mathcal{G},\mathcal{S}}$, relative to source node $\mathcal{S}$ for multicast group $\mathcal{G}$, the router computes the set of downstream interfaces $\mathcal{D}_{\mathcal{G},\mathcal{S}}$ to which it will sequentially send P:

$$\forall P \leftarrow \mathcal{U}_{\mathcal{G},\mathcal{S}}, \textsc{Send}\ P \rightarrow d, \forall\ d \in\ \mathcal{D}_{\mathcal{G},\mathcal{S}} \tag{4.1}$$

Albeit apparently trivial, the operation defined by (4.1) meets several difficulties when it is applied across a wireless network.

In wired networks, a router's interface provides access to either a specific router or to a subnet. In turn, in a wireless network in which radio broadcast links are used, an interface is usually used to cover all the nodes that are located within the node's radio transmission range, regardless of whether they are end-nodes or routers, and regardless of whether the routers are located downstream or upstream in the multicast tree, or whether they are situated on pruned branches, making the forwarding rule in (4.1) ambiguous.

In case nodes employ a single radio module (say radio a), this module is used to provide for the upstream interface. When a node uses two radio modules (say a and b), radio module a can be employed to realize links along the multicast tree; radio b is often used for host access purposes, or to collect packets from source nodes. In the latter case two situations are possible: *i)* the multicast packet P is received on b (the end-node's interface) which means that the source $\mathcal{S}$ is attached across this interface; the received packet is forwarded by module a (if the interface has not been pruned); or *ii)* the multicast packet P is received by module a; Since a is the upstream receiving module, the only radio module that can forward the packet is module b, so that the multicast delivery process is interrupted. Under this scenario, the packet P is able to reach at most the second level of the tree.

This problem is caused by having the interfaces along the wireless network correctly (from the PIM and multicast point of view) matched among the

routers' interfaces and the proper subnets. To solve this issue, we propose to properly define the concept of 'virtual interfaces', and configure them by having them serve to provide proper mapping between router interfaces and designated subnets. We then operate the PIM protocol scheme by using virtual rather than physical interfaces. This is described as follows. Let a virtual *Wired Equivalent Interface* (WEI) be defined as the pair {<physical interface>; <router/subnet>}. A WEI is the interface used to reach a single neighboring (i.e., reached within a single hop) router, or a group of end-nodes residing in the same IP subnet in the wireless network. In this manner, a wireless node with one interface, N neighboring router nodes and M subnets, will configure $N + M$ WEIs. Notice that in a mobile or in general a dynamic mesh network the number of WEIs must be continuously adapted to the changing physical topology of the network.

This definition of WEI resolves the above mentioned problem, since it allows routers to properly classify upstream and downstream WEIs, as illustrated in Fig. 4.1. WEIs retain the same properties (w.r.t. PIM) as those displayed by the use of wired interfaces, so that upstream an downstream WEIs can be used as the standard interfaces for operating in accordance with the PIM process. The associated introduced overhead rate is noted to be negligible. However, without the use of proper countermeasures, such an operation will introduces additional traffic, because each node will separately transmit a multicast packet P across each one of its downstream WEI. The latter packet transmissions are sent and received across a single physical channel. This overhead may lead to traffic congestion, channel saturation, increased collision rates and transport quality degradation. To avoid creating such excess overhead, we dynamically cluster WEIs into $\mathcal{D}_{\mathcal{G},\mathcal{S}}$ oriented groups based on their physical interface, and then instruct the router to transmit only a single copy of P across the designated group. In this manner, we exploit the broadcast nature of the wireless medium, making also use of the

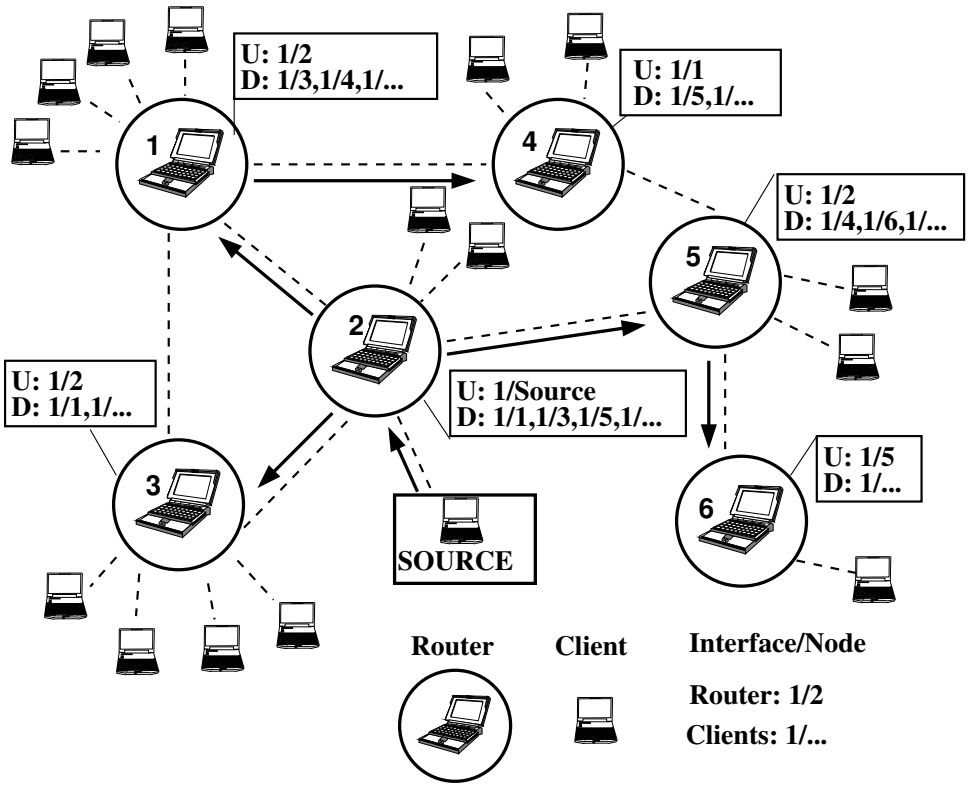

Figure 4.1: Matching WEIs to routers and IP subnets for single physical interface.

structure of the IP multicast process. Such an implementation is readily performed by adding a few program lines to the code that defines the operation of the Wireless-PIM-DM scheme. No change is induced in the actual definition or scope of the protocol's multicasting delivery mechanism.

Another challenge is represented by the network's topological layout. Wired networks are generally static, so that the lifetime of interfaces is long and statically mapped to subnets (unless router failures occur). On the other hand, due to nodal mobility or link degradation events, this is not the case in wireless networks. The resulting topology is continuously changing and may contain nodes, or clusters of nodes, that are temporarily isolated either from the entire network or from specific routes.

Isolated nodes may not be able to receive multicast packet transmissions. To determine the impact of changing topological layouts and nodal mobility

on the multicast operation of the PIM-DM protocol, experimental and/or simulation evaluations must be carried out. The identification and possible resolution of induced issues and multicast distribution degradation phenomena depend also on the underlying unicast routing protocol that is employed (e.g., on the speed at which a link failure is detected and eventually recovered) and on the involved scenario (e.g., including the frequency of topology change and the characteristics of involved isolated nodes).

4.3 Wireless-PIM-DM Performance Metrics

The notation we use in this chapter is reported in Table A.1. We use the following metrics to assess the performance efficiency of the Wireless-PIM-DM protocol scheme:

Control messages traffic We measure the control traffic rate generated by configuring and employing WEIs, exploiting the broadcast properties of the wireless channel. We monitor both transmitted and received message flows, for both control and data traffic, computing the involved control to data message ratio. The involved control message is set as

$$H_m(j) = \frac{1}{T}\int_{t=0}^{T} h_m(j,t)dt \tag{4.2}$$

$$\overline{H_m} = E[H_m(j)] \tag{4.3}$$

where $h_m(j,t)$ designates the control traffic level involving either transmitted or received packets processed at PIM-router j at time t, while T represents the duration of the scenario that is simulated. Eq. (4.3) defines the average control message rate, averaged over all involved PIM-routers.

Convergence time The protocol process is said to converge when the process that involves the distribution of all PIM Prune/Join messages reaches steady

state. Clearly, the protocol continues to maintain the tree and dynamically adapt it to changes the topology or in the identity of PIM-clients. Assume that the protocol convergence rate is faster than nodal mobility rate. Steady state is reached when PIM-routers stop exchanging Join/Prune messages for a suitable period of time T_{ss}. This period is defined as a function of the PIM-DM timeout, i.e., the idling time after which the tree is rebuilt and Join/Prune messages are sent again:

$$\tau_c = \min\{t \,|\, \forall j \in \mathcal{V}, (h_p(j,t,T_{ss}) = 0) \wedge (h_n(j,t,T_{ss}) = 0)\}$$

where $h_p(j,t,T_{ss})$ and $h_n(j,t,T_{ss})$ r represent the number of Prune and Join messages transmitted/received at PIM-router j within a time interval $[t \div t + T_{ss}]$, while $\mathcal{V}$ is the set that consists of the involved PIM-routers. The convergence time τ_c is influenced by the topology, the number of PIM-clients subscribing the multicast group $\mathcal{G}$, and also by the amount of data transmitted by the source $\mathcal{S}$ since PIM-DM is data driven.

Data delivery Defines the fraction of issued multicast packets P that are successfully delivered to PIM-clients. Clearly this metric depends on the topological layout, the number of PIM-clients subscribing to multicast group $\mathcal{G}$, and also on the type and intensity of the multicast traffic flow. In our experiments, we have computed this metric by running a video streaming session originating at a source $\mathcal{S}$ that is external to the wireless network, so that the stream is injected into a selected PIM-router that is acting as a gateway node. We set the streams rate to 1 Mbit/s with constant size chunks of 1200 bytes, so that they fit in a standard Ethernet packet. We assume the source node is not a end-user client, actually it does not belong to the wireless mesh network, but it is directly connected to the single PIM-router with gateway functionality that connect the mesh network to the Internet. Thus, a packet transmitted by the source node is received always by the

same PIM-router, which forwards the packet to its PIM-clients and to its downstream PIM-routers, in accordance with the forwarding rules implied by the multicast tree layout constructed by using the PIM-DM protocol. We have then measured, over the duration of the process, the number of received and missing video chunks, at every PIM-client, as well as accounted for duplicate copies. We have also measured the end-to-end delay levels incurred by delivered data packets. The fraction of chunks received by PIM-client i is computed with using eq. (4.4), while eq. (4.5) is used to compute the throughput rate when averaged over all PIM-clients. Eq. (4.6) and (4.7) are the complement, i.e., the chunk loss rate.

$$R_P(i) = \frac{1}{|\mathcal{C}|} \sum_{P \in \mathcal{C}} r(P) \ , \quad r(P) = \begin{cases} 1 & \text{if } P \text{ is received} \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

$$\overline{R_P} = E[R_P(i)] \tag{4.5}$$

$$L_P(i) = 1 - R_P(i) \tag{4.6}$$

$$\overline{L_P} = E[L_P(i)] \tag{4.7}$$

where P is the generic chunk/packet of the multicast stream, and $\mathcal{C}$ is the set of packets composing the stream. We note that packet duplication is unavoidable on wireless networks, and can also be employed for positive spatial redundancy, but it obviously has a cost in terms of network load. A PIM-client receives normally one or more copies (one from its upstream PIM-router, and others from PIM-routers within reception range that have PIM-clients associated). In addition, we compute Eqs. (4.8) and (4.9) compute the average number of packet copies C_P received by PIM-client i, and the overall level $\overline{C_p}$ when averaged over all PIM-clients. Notice that this measure

is meaningful only when the $R_P(i)$ is very close to 1.

$$C_P(i) = \frac{1}{|\mathcal{C}|} \sum_{P \in \mathcal{C}} c(P) \tag{4.8}$$

$$\overline{C_P} = E[C_P(i)] \tag{4.9}$$

4.4 Wireless Model and Experimental setup

In our experiments, we use IEEE 802.11. The channel data rate is set to 54Mbit/s. We set the transmission power to 16dBm, as commonly used by wireless platforms. Large-scale propagation models have been used to compute the path loss between transmitter and receiver pairs under different channel conditions. Theoretical and measurement based models show that the average received signal power (in dB units) across a link decreases logarithmically with distance. In our experiments, we use the Log-Distance path loss model to model power attenuation as given by eq. (4.10) with the path loss exponent $n = 3.5$, which is suitable for an urban scenario [82]; d_0 is the reference distance (i.e., one meter), and d is the distance between transmitter and receiver:

$$\overline{PL}(dB) = \overline{PL_0} + 10\, n\, log\left(\frac{d}{d_0}\right) \tag{4.10}$$

More details about the parameters used in our experiments are available in [8]. To compute the radio transmission range, we have simulated a simple source-sink scenario where the sink moves away from the source at speed of 1 m/s. The average radio transmission limit for such a configuration is 45 meters. The simulation parameters are reported in Table 4.1.

We have implemented the Wireless-PIM-DM [14] scheme within the ns-3[2] simulator. The software modules developed in this chapter are available at the author's home page or on request. The protocol complies with the standard PIM-DM, and the implementation is modified in accordance with the

[2] www.nsnam.org

Parameter	Value
TxPower	16 dBm
EnergyDetection	-95 dBm
CCA Threshold	-62 dBm
Reference Loss $(\overline{PL_0})$	30 dBm
Path loss exponent n	3.5
Simulation time	180s
Runs	20
Stream rate	1 Mbit/s
Packet size	1200 B
PIM-clients speed	1.4m/s
TxRange	[15÷40] m

Table 4.1: Parameters used for Wireless-PIM-DM experiments.

Wireless-PIM-DM scheme described in Section 4.2 based on the WEI approach.

Furthermore, we have implemented a simple group membership protocol, because the Internet Group Management Protocol (IGMP), including the processes for session setting and end-user joining, protocol has not yet been implemented in the current version of ns-3, The traffic rate generated by such protocol messaging is noted to be relatively negligible.

We used the AODV [78] as the underlying ad-hoc unicast routing protocol. We observe that the proposed protocol would work equally well with other unicast routing protocols, as the latter interact through queries to the unicast routing table.

Each node is either a PIM-router or a PIM-client depending on whether it executes the Wireless-PIM-DM protocol or not, respectively. Indeed, PIM-routers are part of the mesh infrastructure, while PIM-clients are end-user clients interested in the multicast session. We focus on demonstration the operation of our proposed scheme for a static mesh network layout where PIM-routers are fixed in place while PIM-clients may roam over a specified

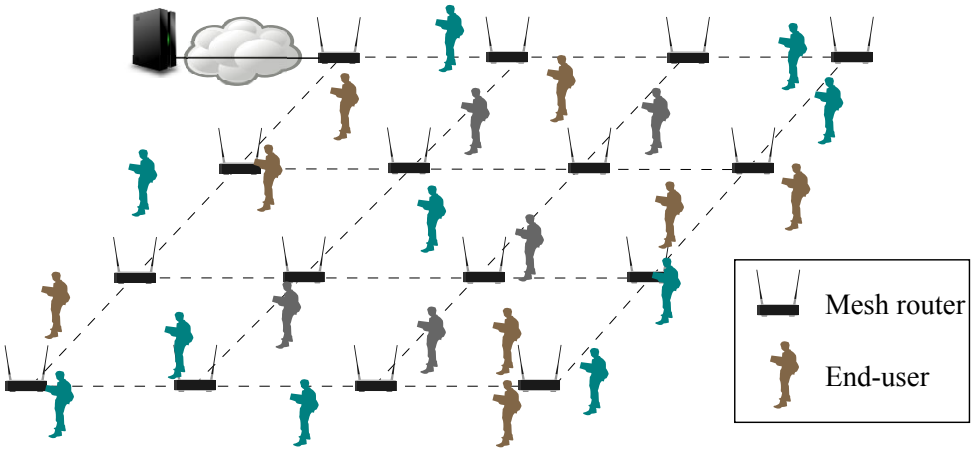

Figure 4.2: Overview of the scenario.

region that is served by these PIM-routers. Future studies will account for the accommodation of mobile PIM-routers.

The wireless mesh network consists of 16 fixed PIM-routers arranged in a 4×4 grid, as represented in Fig. 4.2. The distance between two adjacent nodes in the grid is used as a parameter in our simulation. It is set to be no longer than the radio propagation range (i.e., 45m). The number of PIM-clients is also used as a parameter in our simulation, varying over the range $[1 \div 238]$. PIM-clients are placed randomly within radio propagation range of the PIM-routers. The first scenario is characterized by fixed PIM-clients.

In the second scenario, we use a Random Waypoint Mobility model [44] to model the mobility of a group of PIM-clients over an area which is covered by the installed static PIM-routers. A typical representation of this scenario may be a university campus scenario, where roaming students access the network to view academic news or sports events. Groups of students may move from one area (e.g., a lecture building) to another one at given speed, waiting for a given time (i.e., the pause time) before moving again. We set a group speed $V_{eu} = 1.4\,\mathrm{m/s}$, which represents the speed of an average pedestrian; the pause time $P_{eu} = 40\,\mathrm{s}$, allowing four changes of positions to occur during the simulation run time. Each simulation run lasts for 180 s.

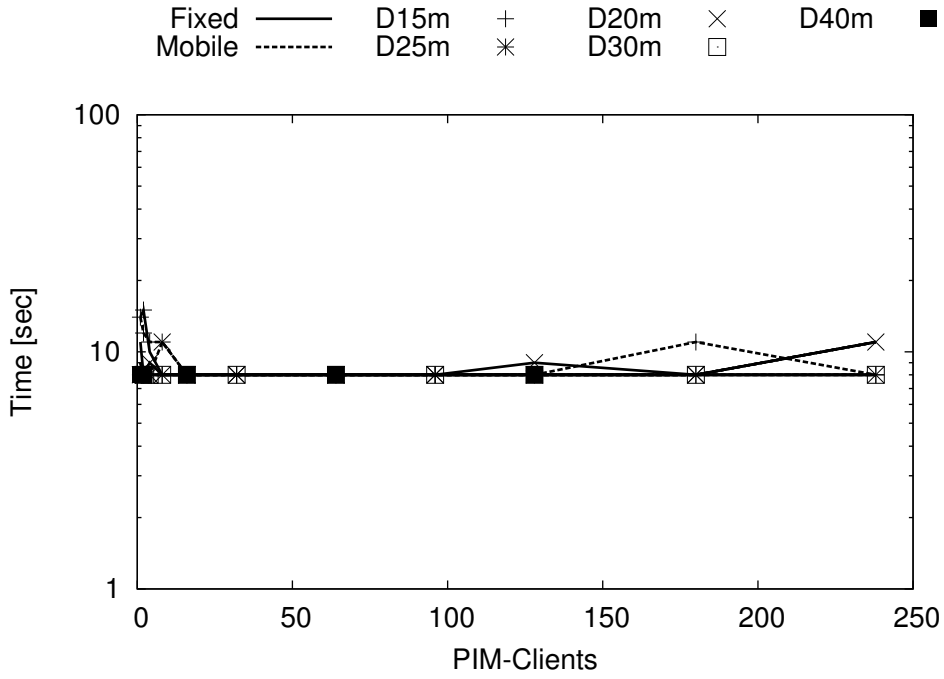

Figure 4.3: Average convergence time τ_c for static and roaming clients.

Performance results have been calculated by averaging over 20 runs, varying seed values; the settings serve to permit convergent behavior and guarantee high confidence levels.

4.5 Results on Wireless-PIM-DM Protocol

In Fig. 4.3, we depict the average convergence time of our protocol scheme, for both the static and roaming scenarios. We observe it to show that the Wireless-PIM-DM protocol converges rather quickly, independently of the number of clients. Although one may expect a longer convergence time for the mobility scenario, this is noted to not be the case. By increasing the number of PIM-clients, we note that a set of several PIM-routers is selected to form a multicast tree, such that the same routers are employed in feeding roaming clients with multicast packets for the complete duration of the session. We observe the average protocol convergence time to be limited to 8s; it increases

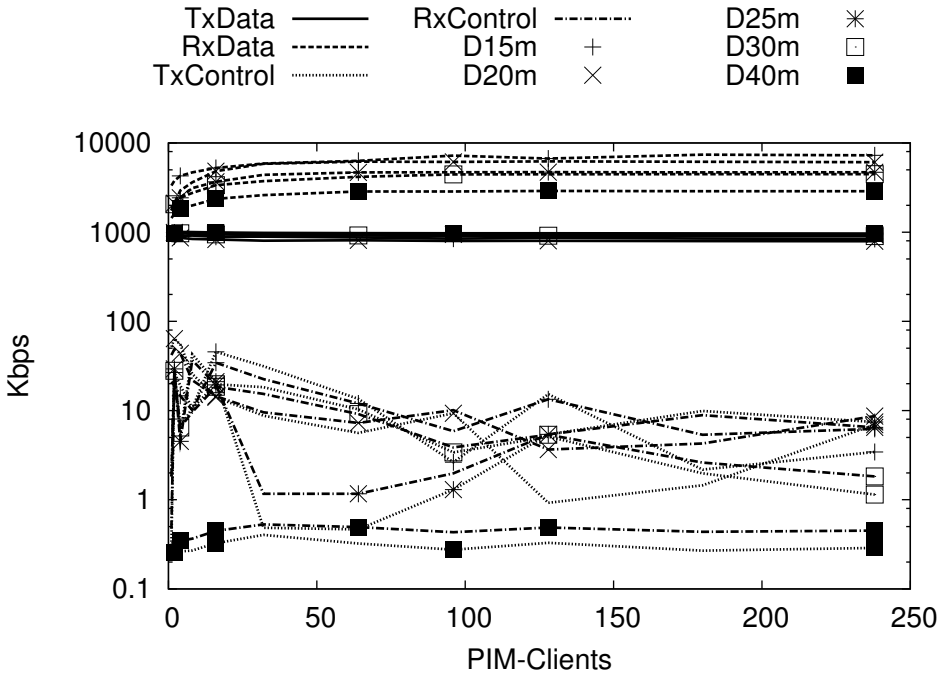

Figure 4.4: Average traffic at PIM-routers with static PIM-clients.

(slightly) to 11s when the network serves a smaller number of clients. In this case, several PIM-routersare pruned.

Fig. 4.4 displays the average transmitted and received traffic levels, including both control and data messages. We observe that the control traffic overhead $\overline{H_m}$ is affected only marginally by the number of PIM-clients, except for the case that involves many fewer PIM-clients. In the latter situation, many PIM-routers issue prune messages and are removed from the multicast tree. Hence, we observe that the control traffic rate level incurred is related to the average level characterizing the distance between PIM-routers. By increasing the grid range used for the placement of PIM-routers, we reduce the ensuing control traffic overhead rate, noting it for the underlying scenario to decrease from 10kbit/s, when the grid range is set to 15m, to 500bit/s for a grid range of 40m. The increase of the distance between PIM-routers reduces the size of the overlapped area, resulting in a reduction of prune and graft messages. We conclude that the resulting PIM-DM associated control

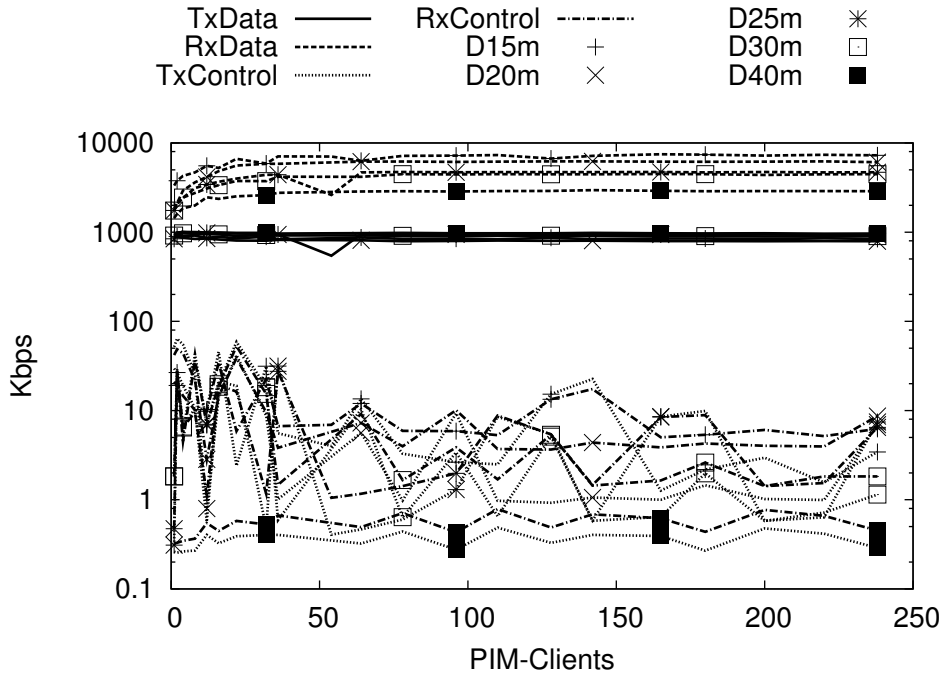

Figure 4.5: Average traffic at PIM-routers with roaming PIM-clients.

traffic rate generated with the incorporated use of WEIs and dynamic grouping does not represent a bottleneck element in the system. Its level is noted to be comparable, if not lower, than the traffic rate generated by a typical unicast ad-hoc routing protocol.

Similar observations are made for the roaming scenario shown in Fig. 4.5. Here, we observe that the control traffic overhead rate is slightly higher than that generated under the static scenario. As they travel, roaming nodes may either activate or deactivate certain PIM-routers, resulting in graft or prune messages that are used for join or leave operations.

Fig. 4.6 shows the average fraction of packets correctly received, missed and duplicated at PIM-clients, under different distance levels between PIM-routers. We conclude that the protocol operation scales well as the number of clients grows, limiting the fraction of missed chunks to a value that is lower than 0.02 in dense networks (i.e., for PIM-routers inter-distance levels of 15m). By increasing the PIM-routers' inter-distance level to a value

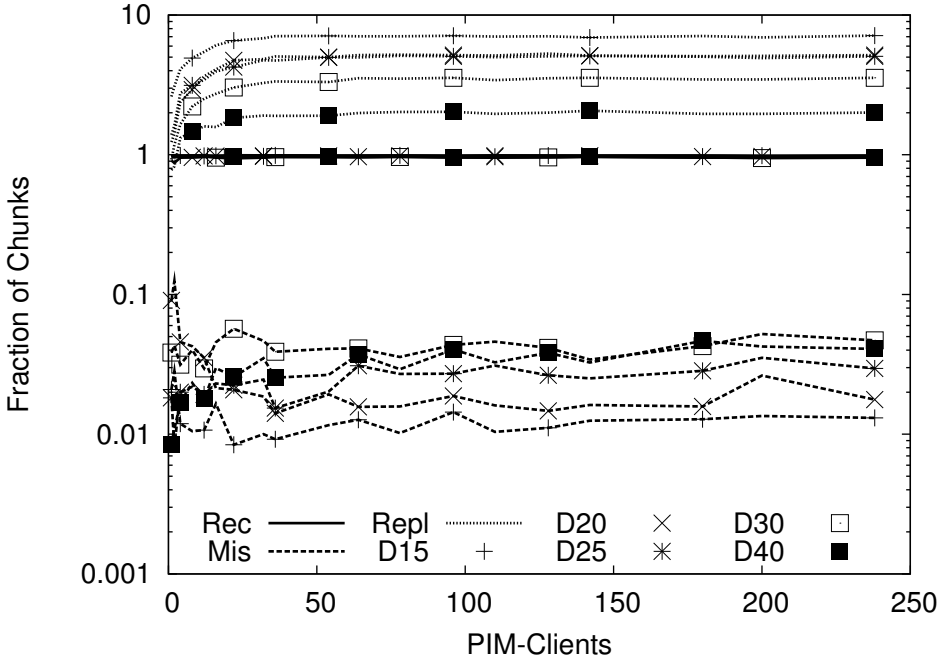

Figure 4.6: Fraction of chunks received, missed, and replicas: static clients.

that is close to the maximum allowed range (i.e., 40m), the average fraction of missing chunks increases slightly, but is still lower than 0.05. The relation between missing/duplicate chunks and the distance between PIM-routers is obvious: PIM-clients are used to recover missed chunks by using the duplicate ones received by other PIM-routers. Thus, (i) short distance values result in a higher rate of duplicate chunks, which reduce the fraction of missed chunks, while the control message overhead increases; (ii) The use of longer ranges results in a reduction in the number of duplicate chunks, leading to an increasing fraction of missed chunks, while reducing the control message overhead (i.e., reducing prunes). The 40m configuration performs better than the 30m one, as explained in the following. By increasing the distance between PIM-routers, we reduce the probability of frame collision events, increasing the spatial diversity element (spatial reuse factor) of the operations. The gap is present until we add more PIM-clients that activate all PIM-routers, making both configurations exhibit similar performance re-

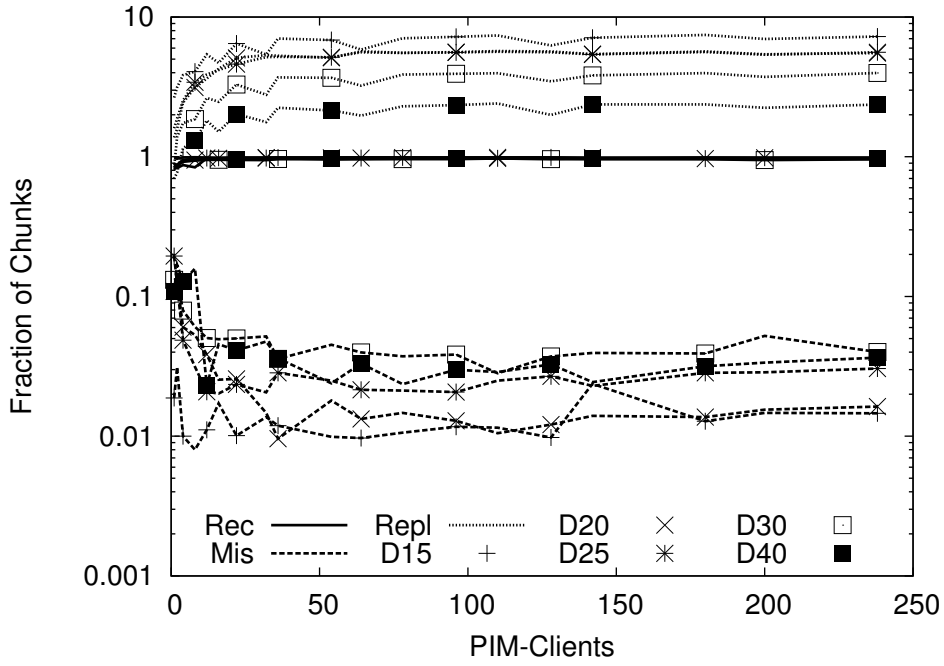

Figure 4.7: Fraction of chunks received, missed, and replicas: roaming clients.

sults. The fraction of received chunks is about 98% (for grid range of 15m) and 95% (for grid ranges of 30m and 40m). As we have expected, the distance between PIM-routers in the grid plays an important role, limiting the number of missed chunks, or increasing the number of replicas. In particular, in a dense grid (i.e., 15m), PIM-clients receive on average about 8 replicas per chunk, while when stretching the grid (i.e., to 40m), the corresponding value drops to about 2 replicas per chunk.

Fig. 4.7 provides the same information for the second scenario, where PIM-clients move within the area covered by PIM-routers. Although we expect a moderate increase of the fraction of missing chunks, due to mobility, this is not observed in results. Rather, we observe a slight decrease to take place in many cases. The reason is that PIM-clients, during their roaming phase, activate additional PIM-routers than the static scenario, and on average, more PIM-routers are involved in the distribution. This increase the number of duplicate chunks received by PIM-clients, resulting in a reduction of the

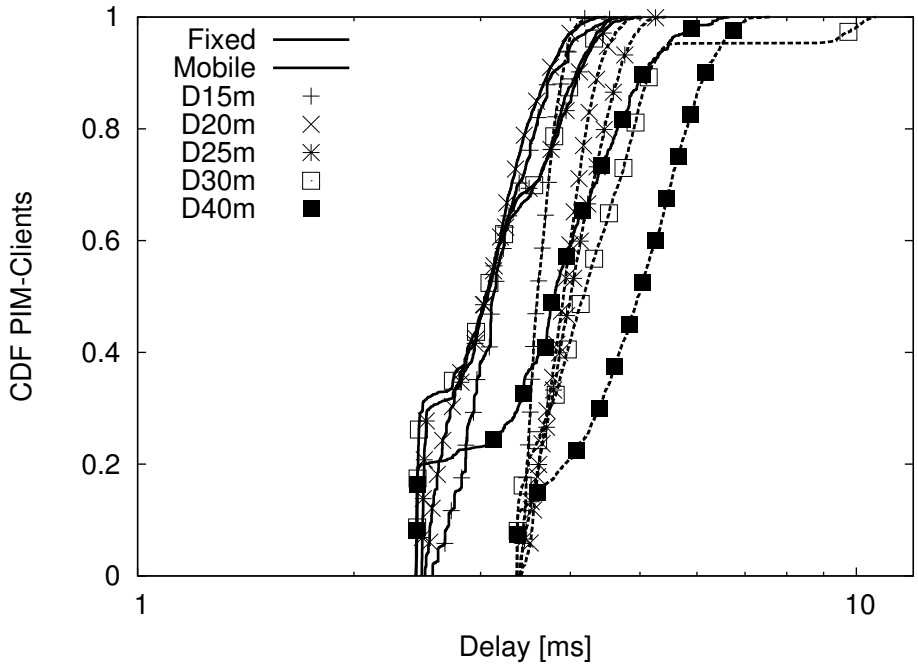

Figure 4.8: CDF of the average chunks delay for nodes that receive at least the 95% of chunks vs. the distance between PIM-routers.

fraction of missed chunks.

Next, we focus on the configuration involving 128 PIM-clients, both static and mobile. The cumulative distribution function (CDF) of the average chunks diffusion delay is shown in Fig. 4.8. We consider only those nodes that have received at least 95% of all the chunks, for both the static and roaming scenarios. As already observed, the grid dimension (distance between PIM-routers) affects the chunks delivery delay, since when it increases, the number of PIM-routers traversed by each chunk, in reaching all PIM-clients, increases too as there is less overlap in the transmission range of PIM-routers. Obviously, PIM-clients that are associated with PIM-routers located closer to the source node receive chunks that have experienced lower delays than those associated to PIM-routers at a longer distance, determining the shape of the CDF. Similar observations hold for the mobility scenario, but with a moderate increase in the chunks diffusion delay than that observed under the

static scenario. This is induced by the mobility of PIM-clients. We note the involved delay level to be lower than about 10ms, so that it does not hamper the distribution effectiveness of real-time multicast traffic flows.

4.6 Discussion

We presented a wireless version of the PIM-DM protocol and its performance in wireless mesh networks that employ ad-hoc routing protocols. Our proposal slightly modifies the wireline version of the PIM-DM protocol through the introduction and use of Wired Equivalent Interfaces (WEIs). In this manner, we avoid the ambiguity involving the operation of upstream and downstream interfaces that will otherwise occur and prevent a PIM protocol to properly operate in a wireless network environment.

We show that the multicasting operation under the Wireless-PIM-DM protocol is scaling well with the number of clients. Through the use of an illustrative video streaming session, we demonstrate the underlying system to yield effective throughput and packet delay performance behavior.

4.7 Peer-Assisted Wireless Multicasting

P2P protocols have been extremely successful for data dissemination, but also for real-time streaming [42, 50, 94]. P2P approaches, however, consume a lot of last-leg transmission resources, which, in case of wireless access, is a problem, especially in light of the intrinsic broadcast nature of the wireless channel.[3]

A successful IP-based streaming system for mobile devices must make the most out of every piece of the network to be efficient and successful. We focus here on the efficient distribution of a multicast stream (typical applications

[3]Part of this work was published in the proceedings of the 10th Annual Conference on Wireless On-demand Network Systems and Services (WONS'13), Banff, Alberta, Canada [7].

range from TV to gaming to context-aware services in push) on an 802.11 access wireless network, considering both the single BSS (Basic Service Set) case and a more complex wireless mesh network.

In this scenario, the key features to take into account are four:

1. The wireless channel is intrinsically broadcast, but reception conditions are not identical for every station, i.e., a broadcast communication may be correctly received by some stations and not by some others;

2. Modern radio technologies are multi-rate, i.e., the better the channel conditions the faster the transmission rate;

3. Broadcast and Multicast transmissions are sent at one chosen rate from the basic rate set (BRS), usually lower than unicast rate, to increase the number of stations that correctly receive the packet;

4. There is a definite trend in enabling direct communication between stations that are within radio range of one another, trying to exploit spatial diversity; 802.11e/n already implement direct stations communications also in infrastructured mode, while 3G+ and LTE are adopting it with the WiFi Direct standard.

Exploiting these four key observations, we propose a P2P recovery protocol, which exploits direct unicast transmissions to retrieve damaged or lost data packets sent either in broadcast or in multicast by the source node.

The key idea is simple: whenever a station receiving a multicast stream misses a packet, it tries to retrieve such a packet from its neighbors stations within its radio communication range. Implementation and the performance achievable are instead far less obvious and require the definition of how multicast is implemented in the access network, the peer discovery protocol, and an intelligent way to identify the best station to try to recover the packet from. We call the overall system PullCast.

4.8 Related Works

The first issue at stake to implement PullCast is solving the problem of multicasting in mesh networks. As pointed out in [60] multicasting in mesh networks can be difficult, specially extending protocols designed for wired networks. In [9] we have proposed and implemented a modification to PIM-DM [14] which can be used in mesh networks, and we will use it in this work. However, any multicast routing protocol can be used.

We do not claim that the idea of cooperative recovery of missing data is entirely new, as cooperation has been proposed in many works to improve performance. For instance, in [84] a Cooperative P2P Repair (CPR) is proposed to improve the reliability of wireless broadcasting via 3G networks, recovering lost packets using 802.11 in ad-hoc mode, while a very recent contribution named BooSTER [101] proposes to enhance performance of DVB-T by recovering damaged data adopting a CPR paradigm where unicast pull messages are sent over the Internet to recover lost information. Other examples on heterogeneous networks exists as well; however, the specific environment of Internet-based multicasting plus local P2P recovery in a 802.11 wireless access has never been studied to the best of our knowledge. In [2] we explored the fundamental properties of hybrid push/pull systems in wired networks, but in wireless networks the protocol has to deal with different constraints and networks conditions.

In [72], Majumda et al. propose a solution based on data redundancy that combines a hybrid automatic repeat request (ARQ) algorithm and Forward Error Correction (FEC) to address the problem of real-time video streaming over WLANs for unicast, while for multicast progressive video coding based on MPEG-4 FGS is combined with FEC.

In [83] Raza et al. explore the benefit of cooperative out-of-band peer-to-peer repair (CPR) to enhance the reliability of wireless multimedia broadcast-

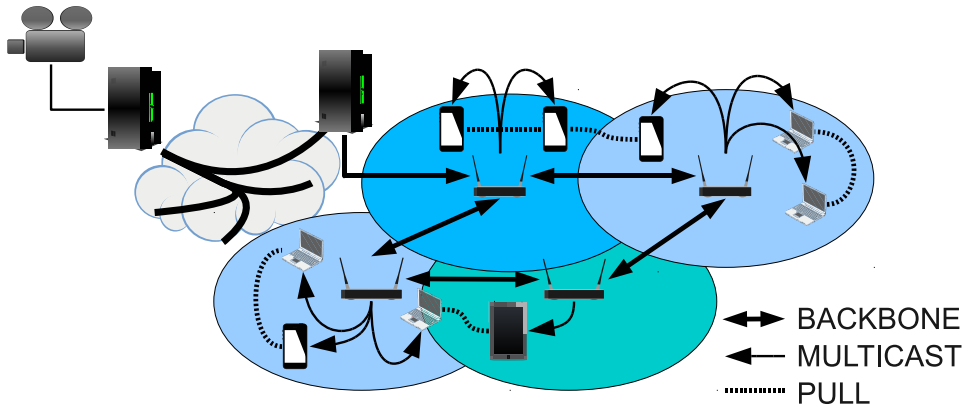

Figure 4.9: Abstract representation of the system where PullCast operates.

ing in hybrid cellular/WLAN networks. In particular, they consider a fully distributed CPR protocol where multimedia content is broadcasted through the 3G cellular network, while lost data packets are recovered over IEEE 802.11 ad hoc P2P network.

In [106] Xiong et al. propose PeerCast, a centralized cooperative system for 802.11-based WLANs. The key observation is that the wireless channel capacity is highly deteriorated by by lower data rates, generating congestion. In PeerCast, the AP (i) pushes the data packets to a subset of clients at higher data rate, and (ii) selects a few clients in strategic positions to relay packets to the other clients at higher data rate, exploiting higher client-to-client Signal-to-Noise Ratio (SNR) and spatial diversity, as well.

4.9 System Overview

We consider a system as depicted in Fig. 4.9. A source node (e.g., TV broadcaster) streams a multimedia service in the Internet. The stream gets to a gateway that distribute it into the wireless access network. Within this latter, the multicast PIM-DM, modified as described in [9], is used to deliver the stream to all the Mesh Nodes (mesh-nodes) involved in the distribution.

Each mesh-node is responsible for one Basic Service Set (BSS), serving end-user; mesh-nodes form a backbone network and also serve end-users stations using standard 802.11e/n. The final hop toward end-users exploit data-link broadcast.

We observe that the number of wireless network interfaces available on mesh-nodes affects the system performance significantly. In particular, in case mesh-nodes have only one interface, packets must be forwarded as broadcast frames on the single interface towards both end-users and other mesh-nodes, that will share the same wireless channel: the backbone and the end-user network will interfere each other, resulting in a system bottleneck. In presence of two (or more) wireless interfaces, instead, one will be used for connecting all the mesh-nodes, and the second will be used for communicating with end-users. Moreover, different mesh-nodes' BSSs can overlap, and end-users stations are scattered in the area served by the mesh network, possibly moving. End-users receive the same video stream, as in TV broadcasts, on-line gaming and similar applications. Different streams can be supported on different multicast groups, albeit resources can severely limit the number of video streams that can be distributed.

We observe that losses among end-users are in general uncorrelated, because they depend on the position-specific channel state. Moreover, the identification of the missed packets is straightforward, because packets are timestamped, sequential and numbered: missing packets are identified immediately upon receiving correctly a packet with a higher sequence number. In addition, video streaming applications may be able to tolerate a limited amount of lost data packets; therefore, end users nodes do not need to receive all packets, but a small percentage of losses is acceptable, and depending on the error correction techniques, the losses percentage might be 5%, to guarantee a good Quality of Experience (QoE), or even 10% with lower quality.

We assume that video chunks are mapped to single packet for the sake of

easy implementation of multicast, so these two terms are used interchangeably in the rest of the chapter. In PullCast, end-user stations missing one or more packets will try to ask these packets to stations which are receiving the same stream and that define a local neighborhood based on the exchange of sporadic hello messages. The goal of PullCast is the definition of an efficient, low overhead protocol to discover if such a situation exist and in this case create neighbor relationships and define an efficient recovery protocol with high chances of success.

4.10 PullCast Design

The key idea is to build a local P2P overlay among the nodes that receive the same stream. The application source node numbers and timestamps each stream packet that can thus be recovered individually ('packet' and 'chunk' are used interchangeably). Packets are sent on a multicast group, reaching the gateway of the wireless access network, then, the gateway forwards such packets towards the mesh-nodes, which also are multicast enabled, feeding the end-users attached in the wireless network. However, being multicast, packets can be easily lost both between mesh-nodes and in the last hop.

PullCast tries to recover lost data packets via unicast transmissions. In particular, each end-user station tries to pull such packets from a one-hop neighborhood maintained for the purpose trying to identify nodes that are as close as possible to exploit high data rates. Packets that are not received (either in multicast or pulled) within a delay δ from their timestamp are lost.

PullCast has to discover node's neighbors that are running PullCast on the same multicast session. The P2P overlay is local to the wireless coverage of each end-user station. If all mesh-nodes serve end-users stations

Algorithm 6 NeighborhoodManagement

```
1: function SendHelloBroadcast(Subnet)//Send a broadcast hello every h_t
2:     HelloMsg ← Node's metrics
3:     SendMsgBroadcast ← (HelloMsg, SNR)
4: end function

5: function RecvHello(Sender, HelloMsg) //Hello message was received
6:     NeighborhoodUpdate (Sender, HelloMsg, SNR)
7: end function

8: function NeighborhoodUpdate(Sender, HelloMsg, SNR)
9:     n = FindNeighbor(Sender, N_j)
10:    UpdateNeighborInformation (n, HelloMsg, SNR)
11: end function

   //Invoked whenever a neighbor is selected at node j
12: function Purge(N_j) //Invoked whenever a neighbor is selected at node j
13:    Expire = h_l × h_t
14:    for all n ∈ N_j do
15:        if TimeToLastContact(n) > Expire then
16:            remove n from N_j
17:        end if
18:    end for
19:    PeerSorting (Policy)
20: end function
```

on the same channel, then the end-users stations can receive the multicast traffic from different mesh-nodes, otherwise the P2P overlay is limited to the single BSS. In both cases, PullCast simply uses broadcast Hello messages that announce the willingness of the station to participate in PullCast. Hello messages are sent every h_t s. Algorithm 6 describes the procedure used by each peer. The Hello protocol is as simple as it can be: nodes periodically advertise themselves by broadcasting Hello messages; each node, upon receiving Hello messages, builds and maintains its neighborhood. Hello messages are not ACKed (they are broadcast frames) and

Algorithm 6 NeighborhoodManagement

```
    //Peer selection for pull messages at node j
21: function PeerSelection(Policy)
22:     Purge(N_j);
23:     if Policy == Random then
24:         n = PeerSelectionByRandom(N_j)
25:     else if Policy == Snr then
26:         n = PeerSelectionBySNR(N_j)
27:     else if ... then
28:     end if
29: end function

    //Invoked whenever a neighborhood changes at node j
30: function PeerSorting(Policy)
31:     if Policy == Random then
32:         n = PeerSortingByRandom(N_j)//Uniform PDF
33:     else if Policy == Snr then
34:         n = PeerSortingBySNR(N_j)//Weighted PDF on SNR
35:     else if ... then
36:     end if
37: end function
```

a neighbor is deleted from the local list after a timeout (i.e., $h_l \times h_t$s) without receiving Hello messages. Furthermore, the neighborhood is limited to $|\mathcal{N}|$ nodes, which are ordered by the Signal-to-Noise Ratio (SNR) of their Hello messages. The on-air overhead of building the P2P overlay is given by the Hello messages only, which are small broadcast data frames. Hello messages need not to be sent frequently as neighborhood changes are related to terminal mobility.

The rationale behind PullCast is to complement the standard multicast delivery by recovering the fraction of lost data packets, as long as this fraction is reasonably small. Nodes compute the chunk ratio K_ω of the multicast packets received within a window of P_ω packets. The K_ω includes the chunks received in multicast by the AP or mesh-node and those recovered using the pull mechanism. An excessive use of pull unicasts (to recover a packets when

Algorithm 7 PullCast

```
    // Called whenever a chunk i is received at node j
 1: function ChunkReceived(Sender, C_j)
 2:     if C_j was pulled then
 3:         CancelPullTimer()
 4:     end if
 5:     ChunkBuffer ← C_j
 6:     C_pull = ChunkSelection (OldestMissedChunk)
 7:     K_ω= ComputeRatio(P_ω)
 8:     if PullTimer ≠ Running AND K_ω ∈ I_P then
 9:         invoke PullLoop
10:     end if
11: end function

    // Called whenever a pull is received
12: function PullReceived(Sender, C_j)
13:     if C_j ∈ ChunkBuffer AND C_j is valid AND P_s then
14:         invoke SendChunk(C_j)
15:     end if
16: end function
```

too many are lost) would simply overload the channel, making the system collapse. From this observation, we define a pull activation interval $\mathcal{I}_{\mathcal{P}}$, limiting the pull recovery mechanism to nodes having $K_\omega \in \mathcal{I}_{\mathcal{P}}$. Therefore, stations receiving a poor quality from the multicast delivery will not use the pull mechanism to recover lost data packets.

PullCast tries to recover the missing packet k closer to its playout deadline δ_k (oldest packet still useful). Algorithm 7 shows the pseudocode of the PullCast protocol. Each node keeps the statistics of the SNR of all the neighbors from which it receives Hello messages. The neighbor to pull from is chosen in the set of the $|\mathcal{N}|$ neighbors with the higher SNR following a weighted random procedure derived from the one we studied in [5], where peer selection was based on nodes round trip time delay (i.e., RTT). Let SNR_i be the SNR of neighbor i at node j, then each neighbor i is selected

Algorithm 7 PULLCAST

```
    // Pull loop to retrieve missed chunks
17: function PullLoop
18:     C_pull = ChunkSelection (OldestMissedChunk)
19:     while PullAttempts(C_pull) ≥ P_m do
20:         ChunkSkipped(C_pull)
21:         C_pull = ChunkSelection (OldestMissedChunk)
22:     end while
23:     K_ω= ComputeRatio(P_ω)
24:     if C_pull > 0 AND K_ω∈ I_P AND Valid(P_s) then
25:         N_pull = PeerSelection (SNR)
26:         if N_pull ≠ NULL then
27:             invoke SendPull(C_pull)
28:             StartPullTimer(PullLoop, P_t)
29:             AddPullAttempt(C_pull)
30:         end if
31:     end if
32: end function
```

with probability

$$P(i) = \frac{\mathrm{SNR}_i}{\mathrm{SNR}_{tot}} \tag{4.11}$$

$$\mathrm{SNR}_{tot} = \sum_i \mathrm{SNR}_i, \quad i \in \mathcal{N}_j \tag{4.12}$$

where SNR_{tot} is simply the sum of all SNR_i. As many studies on ad-hoc networks (e.g., [106, 45]), we use the SNR as a measure of the point-to-point channel quality, furthermore we assume that the SNR measured on (broadcast) hello packets is representative also for higher rates unicast packets.

In video streaming, packets are sent at regular intervals, thus if $\overrightarrow{T_S}$ s is the inter-arrival time of multicast packets, known to all clients, the pull packets asking for missing chunks are sent in an interval $\mathcal{P}_s < \overrightarrow{T_S}$ immediately following the reception of a multicast packet; in addition, each node satisfies at most $\mathcal{P}_r$ pull request within each $\mathcal{P}_s$. A pull that is not served within the pull timeout $\mathcal{P}_t$ is sent to another neighbor for at most $\mathcal{P}_m$ times, satisfying $\mathcal{P}_t \times \mathcal{P}_m < \mathcal{P}_s$, then the packet is marked as discarded.

The protocol is very light, stateless and stable, however its performance is difficult to predict. Given that the inter-time h_t between HELLO messages is very large compared to $\overrightarrow{T_S}$, it is not possible to distribute up-to-date information about the chunks nodes own, so that the use of optimal, informed strategies (see for instance [12]) for chunk selection is impossible, and request are sent blindly. Albeit the scenario is different, the seminal work in [20] set an upper bound to the performance of blind-pull protocols, which means that we should not expect the protocol to be able to retrieve 100% of missing chunks. The attempt of a theoretical modelling is beyond the scope of this work; however, the analysis of the influence of $\mathcal{P}_t$, $\mathcal{P}_m$, $\mathcal{P}_s$, as well as their interaction with standard 802.11 parameters ad the number of retransmission attempts for frames lost at the MAC layer will give interesting insight in distributed protocols attempting to improve multicast performance in wireless access and mesh networks.

4.11 PULLCAST: Performance Metrics

PULLCAST has been designed to enable end-users to recover missing multicast packets to improve the final quality, and it is not meant to substitute multicast itself. The metrics we selected to evaluate its performance are the following.

Video delivery ratio The video delivery ratio is the ratio between the number of chunks received by end-users and the number of chunks in the stream. It considers both the chunks obtained via multicast and those recovered in P2P mode.

$$\mathcal{C}(j) = \frac{\sum_{i=1}^{|\mathcal{C}|} r(\mathcal{C}_i)}{|\mathcal{C}|}, \quad r(\mathcal{C}_i) = \begin{cases} 1 & \text{if } \mathcal{C}_i \text{ is received} \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

Video delivery delay The video delivery delay measures the average chunks diffusion delay. We define the diffusion delay for the $\mathcal{C}_i$ as

$$\delta_j(i) = T_j(\mathcal{C}_i) - \overrightarrow{T_{\mathcal{S}}}(i)$$

where $\overrightarrow{T_{\mathcal{S}}}(i)$ is the time the source generated $\mathcal{C}_i$, and $T_j(\mathcal{C}_i)$ is the time in which node j receives $\mathcal{C}_i$. The fraction of chunks received via multicast is affected by a uniform delay, while higher delays are experienced by those received in pull.

Channel load The channel load measures the fraction of time each node "see" the channel occupied. The channel is defined as occupied when it is:

1. Sensed busy or occupied by other stations,
2. Used by the node for transmitting or receiving frames,
3. Unavailable because of MAC protocol internals (Inter Frame Spaces, Backoff Procedures, etc.),

The channel status is a property defined at each station, as propagation conditions may lead to different situations in different places. Let $\eta(i) \in [0, 1]$ be the channel load measured at station i, then the average channel load is $\bar{\eta} = E[\eta(i)]$ including in the average also the access point or mesh-nodes. $1 - \eta(i)$ is the fraction of time that the channel at station i is idle, and is the probability of finding the channel available with a random sampling.

4.12 Simulation Model and Methodology

The notation we use in this work is reported in Table A.1. We have implemented PullCast in ns-3 [43]; the source code will be available in the public distribution after completing the code documentation and the approval of the ns-3 board; for the time being it can be obtained from the authors.

Each simulation is run for 160s, and presented results are averaged over 10 independent runs.

The source node streams at a constant packet rate corresponding to $\overrightarrow{T_S}$ = 12ms, for the sake of simplicity we assume all packets have a payload of 1400B. We set the $\mathcal{P}_t$ equal to the $\overrightarrow{T_S}$, i.e., either chunks are recovered immediately after their loss has been identified or they are lost. The $\mathcal{P}_s$ is further reduced by a small time guard, resulting in 10ms. We set the number of pull attempts $\mathcal{P}_m$ and the number of pull reply $\mathcal{P}_r$ equal to 1, as preliminary results indicate that different values tend to overload the channel; the true impact and the optimization of these parameters is left for future work. The simulation parameters are reported in Table 4.2.

We recall that blind pull cannot achieve 100% chunk delivery; however, video streaming applications can tolerate a limited amount of lost data packets, and FEC techniques can be used to compensate these losses. What we aim at is guaranteeing a good QoE, hence, we define a quality threshold Q_{thr}, representing the minimum number of chunks each end-user should receive to experience a good video streaming session.

End-users stations are uniformly distributed within a radius $R_{\mathcal{N}}$m from the AP/mesh-nodes, which are static, while end-users stations might be static or mobile, The number of end users stations served by each AP is 100, while the hello messages timeout h_t is 10s.

4.12.1 Scenario

In this work we focus on a simple scenario where one multicast transmitter (an AP or mesh-node), deliver the stream in a BSS. This allows the evaluation of PullCast in a simple and controlled environment and provides a benchmark for future, more complex scenarios.

The second scenario consists in a *multi-cell* wireless access network with several mesh-nodes. A typical representation of this scenario may be a uni-

versity campus scenario, where students access the network to watch sports events or news. The mesh-nodes forming the multi-cell are equipped with two wireless interfaces; the former is used to communicate with other mesh-nodes, and the latter provides wireless access to end-users stations. In this work we refer to end-user and backbone network to address the mesh-node-to-end-users and the mesh-node-to-mesh-node network, respectively. Thus, the first scenario has only the end-user network, while the second scenario consists of one backbone network and several end-users networks. We assume that end-users can move towards their destinations following a given path, for instance, students moving from one building to another one, following the sidewalk. Thus, in a chain topology mesh-nodes are placed at a fixed distance d, forming a chain, providing exactly two neighbors to each mesh-node.

4.12.2 Channel and Radio Model

We use the Log Distance Path Loss to model the radio propagation model, where the path loss exponent is set to $n = 2.6$, which is a typical value used for urban scenario [82], while the path loss at 1m is equal to 52.0597dB. The radio transmission power is set to 20dBm, while the receiver sensitivity is set to -95dBi: common values for wireless interfaces. In addition, the end-users network is affected also by the Rayleigh Fading Model, resulting in an average AP coverage radius of about 120m. Moreover, the unicast data rate is set to 54Mbit/s, while the broadcast data rate is a parameter of our study. We remark our interest in evaluating the PullCast protocol that acts within the end-users network not the performance of the backbone network; Thus, we assume a reliable mesh network, where almost all data packets are received by all the mesh-nodes, otherwise the PullCast evaluation will be biased by the losses at the mesh network. Finally, we indicate the station short retry count (SSRC) [11] with STA_{SSRC}.

Radio system	IEEE 802.11e
Path Loss Model	Log Distance: $n = 2.6$, $L_0 = 52.0597$dB
Fading Model	Rayleigh (only end-users network)
Antenna Gain/Loss	Access TxGain=RxGain=1dBi
Antenna Gain/Loss	Backbone TxGain=1dBi, RxGain=5.5dBi
Radio	TxPower=20dBm, Sensitivity=-95dBm
Mobility Model	Random Waypoint: $V_{eu} = 1.4$m/s, $P_{eu} = 30$s

Table 4.2: Parameters used for exploring the PullCast protocol.

4.12.3 Mobility Model

A typical representation of the proposed scenarios are university campus, where roaming students access the network to see sports events or university news, or shopping areas, where customers watch news or advertisement on available discounts, while moving towards shops. In our work, end-users move according to the Random Waypoint Mobility Model [51]: each end-user selects a destination point and then moves towards the destination at a pedestrian speed V_{eu} set equal to 1.4m/s. Once the node reaches the destination, it waits for a pause time P_{eu} set equal to 30s; then it repeats the process and selects a new destination point. Each node has a set of four destination points, that are located within a radius $R_{\mathcal{N}}$ from mesh-nodes.

4.13 Simulation Results

First, we analyse the influence of some MAC layer parameters on the P2P overlay and protocol performance. The parameter STA_{SSRC} in 802.11 controls the number of retransmission attempts per frame, before discarding the frame. As 802.11 is extremely sensitive to this parameter and rate fallback algorithms may lead to heavy congestion we want to understand if the standard value $STA_{SSRC} = 7$ is suitable for our application Fig. 4.10 and Fig. 4.11 show the fraction of end users that receive the target Q_{thr} of 95% of chunks

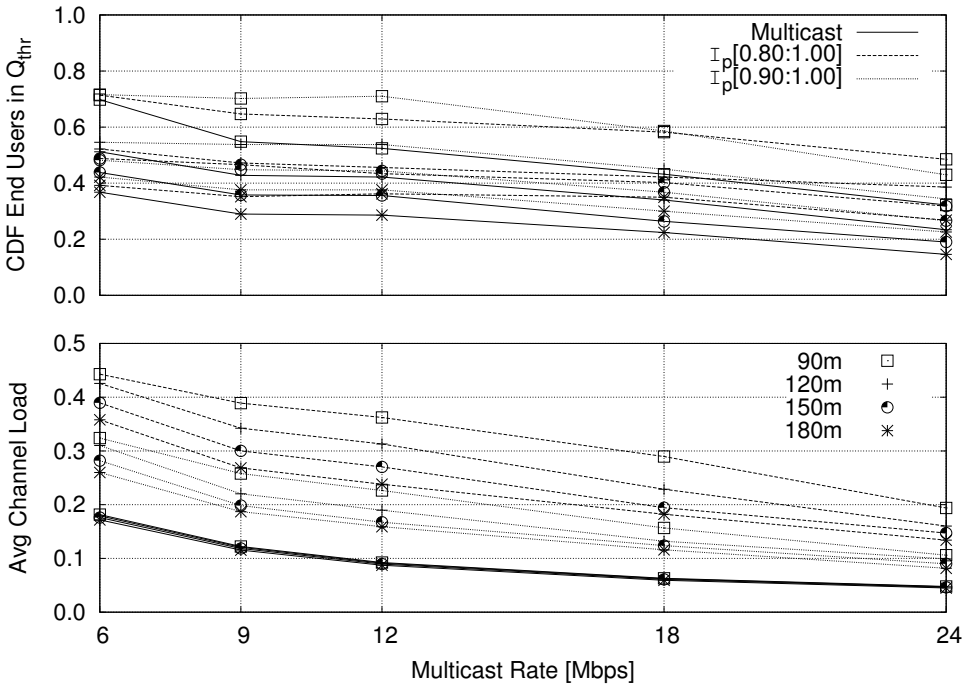

Figure 4.10: End users with $Q_{thr} \geq 95\%$ (Top) and Average Channel Load $\bar{\eta}$ (Bottom) for different BC and $R_{\mathcal{N}}$, with $STA_{SSRC} = 7$.

(Top) and the average channel load experienced by end users stations (Bottom), for different multicast data rate and radius $R_{\mathcal{N}}$, comparing the simple multicast distribution with PULLCAST for different $\mathcal{I}_{\mathcal{P}}$ intervals. Fig. 4.10 refers to the standard $STA_{SSRC} = 7$ and Fig. 4.11 to $STA_{SSRC} = 0$.

A few observations are in order. First, PULLCAST provides improved performance in all cases. Second, for these applications it is much better to avoid MAC layer retransmissions as they simply overload the channel, and do not offer improved performance. Indeed, such retransmission will interfere with the broadcast delivery of the multicast streaming, intensifying the number of collisions, thus, increasing the number of lost multicast data packets. Finally, it is clear that broadcasting the multicast stream at low rates help increasing the delivery ratio, but it also increase the channel load, so that additional background traffic would be penalized (if service differentiation is used) or will interfere with the streaming application. For the remainder of

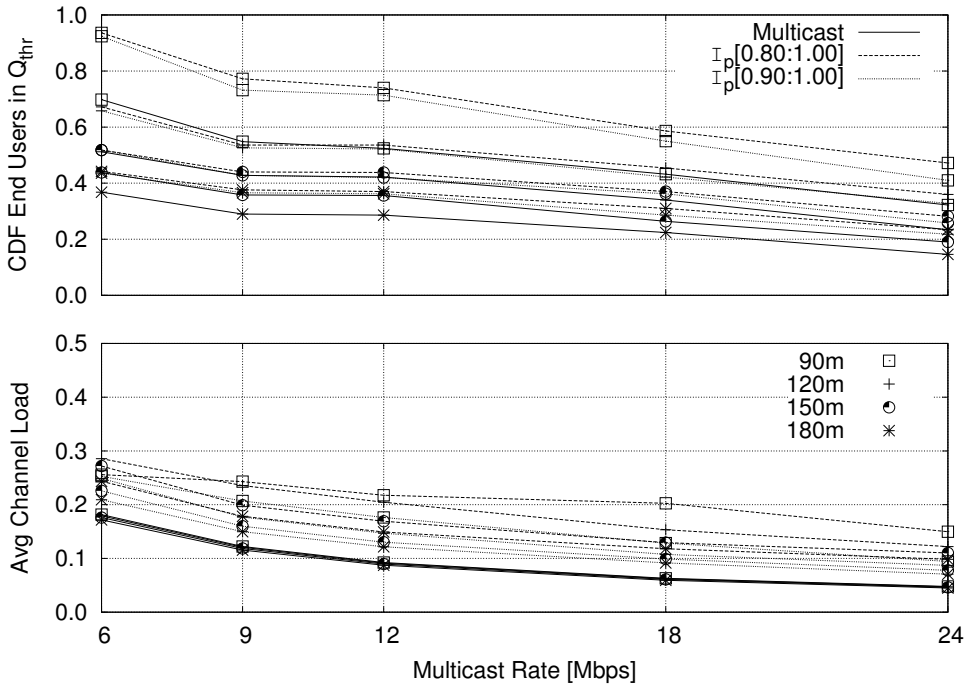

Figure 4.11: End users with $Q_{thr} \geq 95\%$ (Top) and Average Channel Load $\bar{\eta}$ (Bottom) for different BC and $R_{\mathcal{N}}$, with $STA_{SSRC} = 0$.

the work we set the broadcast rate to 18 Mbit/s, both for multicast and for HELLO messages, and we set $STA_{SSRC} = 0$.

Next we focus on the impact of the clients distance, i.e., $R_{\mathcal{N}}$, and also try to gain insight on the impact of the recovery interval of the protocol $\mathcal{I}_{\mathcal{P}}$. Fig. 4.12 reports the entire CDF of delivered chunks to all users, for different values of $R_{\mathcal{N}}$ and $\mathcal{I}_{\mathcal{P}}$ larger than 0.8 and 0.9 respectively. In all combinations PULLCAST improves performance, indicating that the protocol is robust and keep the load of the channel under control even in limit conditions as when the distance of clients from the AP can be as large as 180 m. The highest gain is for $\mathcal{I}_{\mathcal{P}}$ larger than 0.8. Additional results not shown to avoid cluttering the graph indicate that trying to recover the video more aggressively tends to saturate the channel in some conditions, indicating that the protocol is becoming unstable.

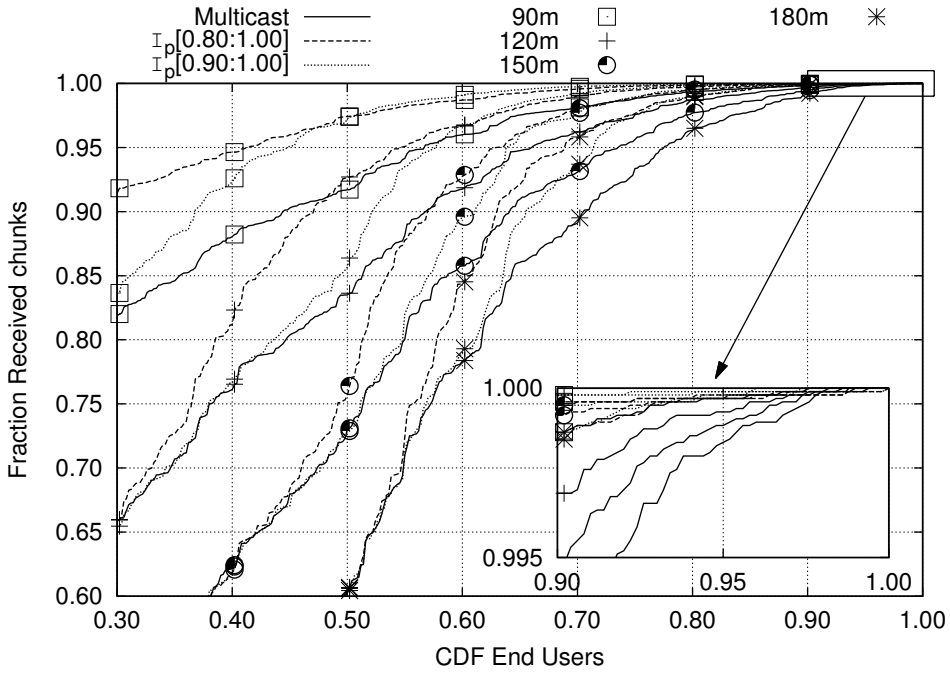

Figure 4.12: Fraction of chunks received for different values of $\mathcal{I}_{\mathcal{P}}$ and $R_{\mathcal{N}}$: $STA_{SSRC} = 0$ and $BC = 18$Mbit/s.

Fig. 4.13 reports the chunks diffusion delay, complementing the information of Fig. 4.12. The size of the $\mathcal{I}_{\mathcal{P}}$ interval influences the chunks diffusion delay: large $\mathcal{I}_{\mathcal{P}}$ intervals increase delays as more chunks are recovered by more peers increasing channel contention. The delay is clearly due to contention, as increasing the coverage area delay decreases, because nodes are less dense, less chunks are recovered as we have seen and in some cases space diversity transmission may happen.

4.13.1 Impact of Mobility

Fig. 4.14 shows the fraction of clients with $Q_{thr} = 95\%$ and $\mathcal{I}_{\mathcal{P}} = [0.90 : 1.00]$ for different values of BC and $R_{\mathcal{N}}$, under mobility conditions. We observe that in dense network, the mobility conditions do not influence our system, sometimes improves the fraction of clients reaching Q_{thr}, as shown for $BC = 6$Mbit/s and $R_{\mathcal{N}} = 90$m, where almost all peers reach the Q_{thr}in

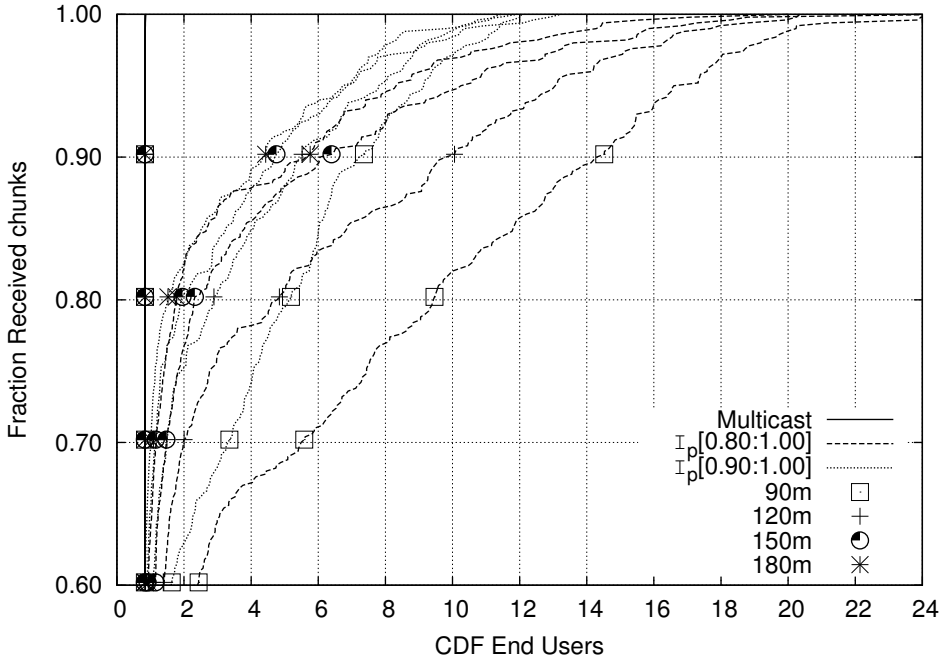

Figure 4.13: Average chunks diffusion delay varying $\mathcal{I}_{\mathcal{P}}$ and $R_{\mathcal{N}}$: $STA_{SSRC} = 0$ and $BC = 18$Mbit/s.

case of all roaming clients; However, if we move towards sparse networks, by increasing either the BC or the $R_{\mathcal{N}}$, or both, we observe that the mobility influence PULLCAST significantly; In particular, the configuration where all peers peers are roaming from one AP to another one, always withing $R_{\mathcal{N}}$, we observe significant performance reduction, while we limit the performance reduction when having half of peers static, for instance, for $BC = 18$Mbit/s and $R_{\mathcal{N}} = 120$m we have about 42% of peers in Q_{thr} for both static and half static configurations, while the full mobile has only 32%.

4.13.2 Extending Results to Single Channel Meshes

We consider a very simple topology with a mesh network extending coverage with four mesh-nodes arranged in a linear topology, and with the very limiting condition of using a single channel for all four BSS (mesh-nodes' backbone is instead on different resources). If BSSs are arranged on different channels,

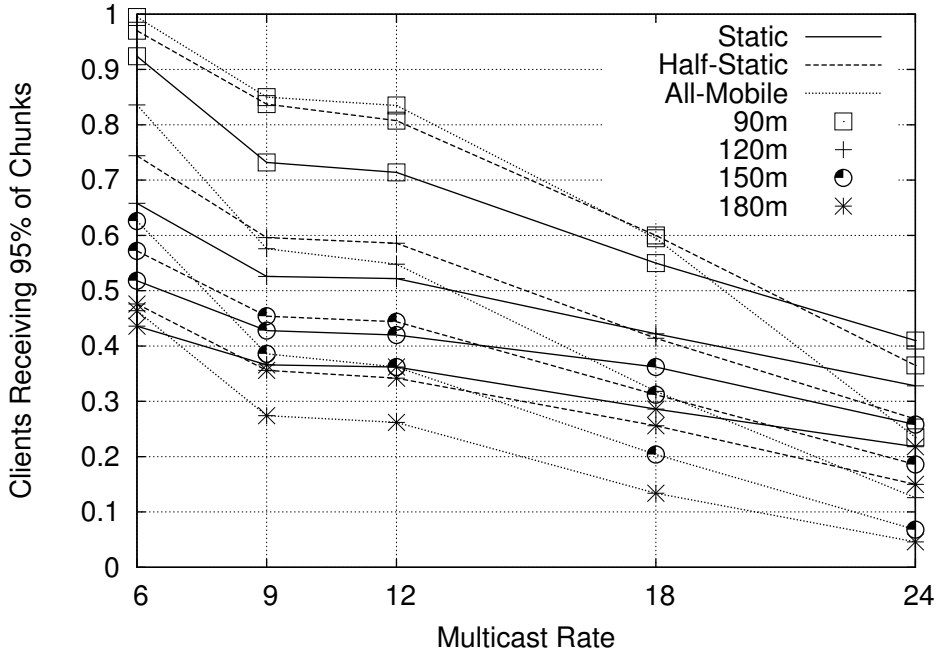

Figure 4.14: End users having Q_{thr}=95% with roaming stations for different BC and $R_{\mathcal{N}}$: $\mathcal{I}_{\mathcal{P}} = [0.90{:}1.00]$.

the situation is not much different from a single BSS, as BSSs are independent one another.

Fig. 4.15 shows the fraction of end-users with $Q_{thr} = 95\%$ and the corresponding average channel load experienced by such stations, for the chain topology. We observe that PullCast outperforms the Multicast delivery for any configuration; for instance, for $BC = 18$Mbit/s, PullCast serves from 4% to 10% more clients than Multicast, while limiting the average channel occupancy. The latter, is lower than the previous scenario (i.e., single cell) because the number of end users stations are uniformly distributed among the four mesh-nodes, resulting in significant reduction of channel contention; however, this condition reduces node's neighborhood, actually limiting the PullCast properties.

Finally, in Fig. 4.16 we analyze the impact of roaming clients for both Multicast and PullCast. We remark that we use the Random Waypoint

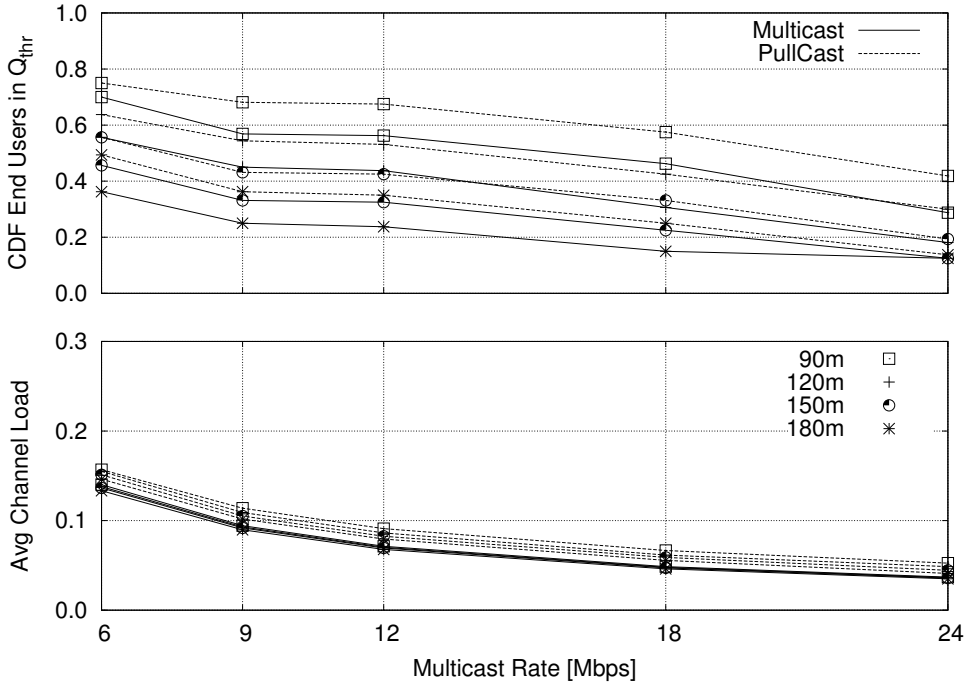

Figure 4.15: End users having $Q_{thr} = 95\%$ (Top) and Average Channel Load (Bottom) in a chain topology for different BC, $R_{\mathcal{N}}$, and $\mathcal{I}_{\mathcal{P}}$ intervals.

Mobility Model, where each node selects a target destination within a radius $R_{\mathcal{N}}$ of a random mesh-node; Moreover, mesh-nodes cells are only partially overlapped, thus, nodes, while moving, might move through some area that is not covered by any mesh-node, and this condition is more evident in sparse networks with lower multicast data rate BC. However, even under this limited condition, again, PullCast is able to serve more clients than Multicast; In particular, when 50% of clients are static, PullCast performs better, since such static nodes provide a good pool on which recover lost data packets; However, in full roaming scenario, as we expect, the number of clients served decreases, because clients might cross cells through that areas that are not well covered: such results suggest to change the design of the current mesh network, reducing the distance between mesh-nodes, or even increase the transmission power.

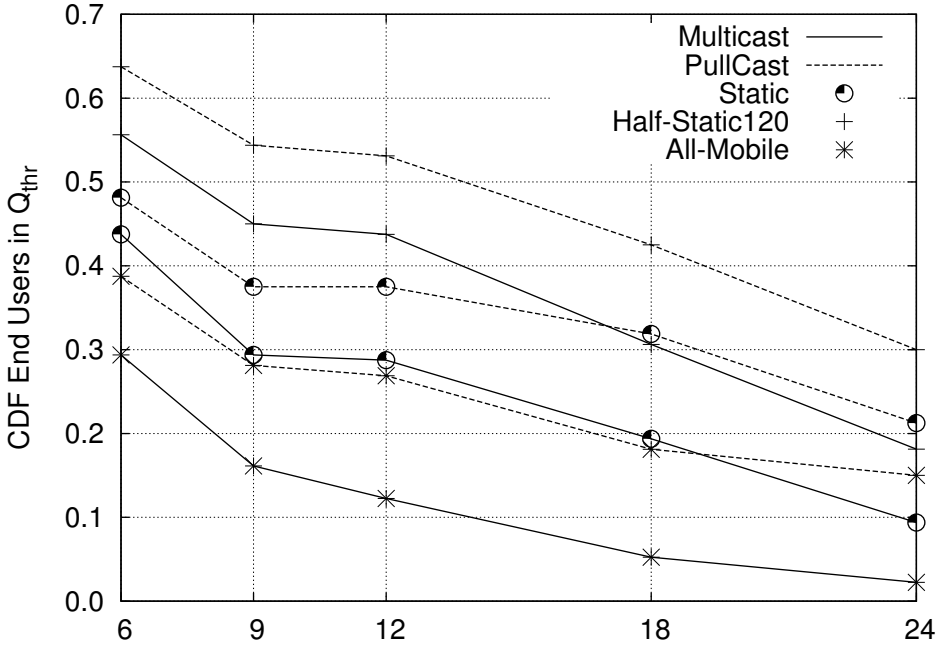

Figure 4.16: End users having Q_{thr}=95% with roaming stations in chain topology with different values of BC and $R_{\mathcal{N}}$.

4.14 Conclusion

In this chapter we present the issues that prevent the PIM protocols to work over wireless mesh networks. We propose the Wired Equivalent Interfaces solution, yielding to a Wireless-PIM-DM protocol version that works properly and efficiently across a wireless ad-hoc network, while requiring only minor modifications to be applied to the commonly employed protocol. In illustrating the performance of a wireless network system that uses the modified protocol, we consider a network system that serves static or roaming users that connect to a wireless mesh network. We show that the multicasting operation under the Wireless-PIM-DM protocol is scaling well with the number of clients. Moreover, we run an illustrative video streaming session, evaluating the corresponding throughput and packet delay performance behavior.

Then, we introduce PullCast, a new peer-assisted video multicasting protocol for wireless mesh networks. The key idea of PullCast is to complement the standard multicast delivery protocol, by activating peer-to-peer recovery mechanism, in order to retrieve a small fraction of lost data packets from nodes' neighborhood. We observe that the average channel load induced by the pull recovery mechanism is limited and lead to significant increasing of peers reaching the target $Q_{thr} = 95\%$, showing that even the chunks diffusion delay is rather limited to 24ms. We discuss the impact of mobility, for different fraction of roaming clients: from all static, to half mobile, to only roaming mobile clients. We observe that in dense network PullCast gains benefit from mobility, while in sparse network, the mobility slightly reduces our protocol performance, which always perform better than Multicast. Finally, we analyze a simple mesh network where mesh-nodes are placed in a chain topology, under both static and mobile conditions, observing that PullCast is able to serve more clients than Multicast, obtaining fairly good results even in case of sparse network where all clients move from one mesh-node to another one.

Regarding the Wireless-PIM-DM protocol, we plan to examine the impact of PIM-routers mobility; in particular, it is of interest to characterize the minimum number of PIM-routers that will have to be activated. We also aim to analyze the impact of limited bandwidth resources and of noisy channel features on the resulting construction process of multicast tree layouts and on multicast delivery, under the use of the Wireless-PIM-DM protocol. In addition, we intend to investigate the impact of using different unicast routing protocols, both flat and hierarchical, on multicast tree construction and on the dynamic convergence behavior of the operation when regulated by the Wireless-PIM-DM protocol.

The PullCast protocol offers several potential future research directions. The first of these is to investigate on peer selection and chunk selection

policies, possibly moving from a blind to a two sided systems, where peers periodically exchange information about their buffer maps, and pull might include a list of lost data packets, not only one as done in this study.

Next, we plan to continue our analysis of multi-cell topologies, extending our work for chain and even grid topologies, investigating on the mesh network design to guarantee high quality of experience to end-users stations.

Finally, PullCast can be improved through a better definition of the wireless transmission model, by including building and obstacles that might either improve or completely block the signal propagation.

Further research include also exploring several streaming rates, as well as better per-node parameters where the protocol adapts itself to node's status, e.g., adapting the $\mathcal{P}_t$ to node's channel experience, maximum number of pull retries and pull replies per pull slot.

Chapter 5

GRAPES: Generic Resource-Aware P2P Environment for Streaming

5.1 Introduction

Peer-to-peer (P2P) technologies are becoming increasingly popular as a way to overcome the scalability limitations intrinsic in traditional network applications based on a client/server paradigm. In particular, there is a great interest in P2P streaming applications (both Video on Demand and TV-like live broadcasting systems), because they have high demands in terms of bandwidth requirements. IP-level multicast could help in reducing the network bandwidth required by an audio/video streaming system, but it is not supported on the Internet.

In the last years the scientific community produced a lot of research work on improving the P2P streaming technologies to provide better quality and to better exploit the available resources; however, commonly used P2P TV applications, such as PPLive, UUsee, and SopCast, do not always follow such trends, and are still based on architectures deriving from file sharing applications, whose requirements are far apart from (live)streaming. As a result, the network bandwidth is not used efficiently, and the P2P TV clients can provide a good user experience only by being very aggressive in network usage.

In the authors opinion, such a lack of technology transfer from the research community to commonly used applications can be solved through the availability of open-source P2P streaming applications providing the researchers with a working codebase that can be easily modified to experiment with and to integrate novel ideas. To target this goal, it is useful to remember how the free availability of open-source kernels as Linux or FreeBSD helped the OS research community! Unfortunately, integrating experimental techniques in an application sometimes requires drastic changes to its structure: for example, changing the local chunk selection algorithm can be easy, but changing a P2P streaming application from "epidemic style" push [12] to pull [42, 109], or more complex strategies [5] can require a complete redesign.

In other words, developing *one single open source application* can impose too many constraints on the research that can be performed on it, requiring to rewrite the application every time a new solution has to be tested (in [66] the authors had to implement 5 different P2P streaming application, writing more than 10000 lines of C++ code).

This work proposes a set of *generic* and *reusable* components forming a codebase to develop P2P streaming applications with (almost) any structure. Such a toolkit, named GRAPES (Generic Resource Aware P2P Environment for Streaming), provides a set of building blocks that researchers can use, combine, and modify to test and compare different solutions fostering the development of new ideas as it happened in OS research [35]. To the authors' best knowledge, similar toolkits do not currently exist in the P2P community. A possibly close work [30] identified a common API for a Key-based Routing Layer that can be used as a base for various P2P services, but no implementation of such an API has been freely released to the research community.

This chapter is organized as follows: Section 5.2 discusses the requirements for GRAPES, which drove the main design choices and influenced the GRAPES structure, described in Section 5.3; Section 5.4 describes how to use

GRAPES, and presents some early experiences (showing how GRAPES can simplify the development of P2P streaming applications); finally, Section 5.5 presents the conclusions and describes ongoing work.[1]

5.2 Requirements

First of all, the functionalities provided by GRAPES should be usable by as many applications as possible in as many different environments as possible. This means that GRAPES should be able to run in many different platforms/operating systems, and should be accessible to many different development tools and runtime environments. For this reason, GRAPES has been implemented as a C library, since almost every development platform provides a C compiler, and it is quite easy to develop bindings to other languages such as python, java, and other high-level languages. In addition, C++ programs can directly link to the GRAPES library without needing any kind of wrapper or special bindings. Moreover, C does not requires complex runtime support, or system libraries.

For the same reason, the amount of dependencies for GRAPES has been reduced to the minimum (e.g., no dependencies on external libraries). The result is that GRAPES can be used on many different systems, ranging from large network servers to small and not-so-powerful embedded devices.

A second requirement, more difficult to fulfill, is that GRAPES should not impose any particular structure to the applications using it, so that the library can be used applying different programming paradigms (ranging from event-based, reactive, programming to thread-based multiprogramming). In this way, GRAPES supports different programming styles (some program-

[1]This work is supported by the European Commission through the NAPA-WINE Project (www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant No. 214412.

Part of this work was published in the proceedings of the 2010 ACM Workshop on Advanced Video Streaming Techniques For Peer-to-Peer Networks and Social Networking, Pisa, Italy [1].

mers prefer thread-based multiprogramming because of its simplicity, while others are against the thread abstraction [95]). This second requirement has some serious implications on the API exported by the library, since concurrency handling and support for parallel activities have to be moved from the library to the application using it. As it will be shown in the following, this has been obtained by removing from GRAPES the code for receiving data from remote peers (and for demultiplexing the received data), and by leaving such a task to the application. The received data will then be passed to GRAPES by invoking an appropriate data parsing function.

A third requirement is modularity: GRAPES should provide all the basic functionalities needed by a generic P2P application, without forcing the application to use unneeded code. For this reason, the GRAPES functionalities have been grouped in several *modules* (that will be described in Section 5.3) and described by a well defined API. Each module has its own API (described by a C header file), and can be used independently from the others (if a module needs the functionalities of a different one, it will use them through the public API, so that each module can be easily replaced by a user-provided implementation). As previously mentioned, all the modules that need to interact with remote peers export a data parsing function (named `ParseData()`, plus a prefix dependent on the module name).

Applications based on GRAPES communicate through *messages*, which can contain data to be diffused or signalling information. Such messages are encapsulated in network packets and sent by using a network abstraction layer, named *network helper*, which allows to easily change the protocol used for transmitting the messages, to port GRAPES to different architectures, and so on. GRAPES modules can directly send messages (by invoking the network helper), or can simply construct them, leaving to the application the responsibility of sending the messages (still through the network helper). On the receiving side, applications are responsible for invoking the network

helper to receive messages, and pass them to the correct GRAPES modules (by invoking the correct `ParseData()` function). This architecture also enables optimization like embedding multiple messages in a single packet to reduce the network traffic.

5.3 Design and Structure

Since GRAPES aims at providing the basic blocks needed for building a P2P streaming application, the most important modules composing such applications have been identified. A preliminary analysis revealed that a generic P2P streaming application usually needs, in addition to the net helper, the following modules.

- A *Peer Sampling* mechanism, providing each peer with continuously up-to-date random samples of the entire population of peers;
- A *Chunk Trading* to send/receive pieces of a media stream (called *chunks*);
- A *Chunk Buffer* to store the received chunks so that they can be forwarded to the other peers;
- A *Chunk ID Set* data type to send signalling information about the received or needed set of chunks;
- Some *Scheduling* functions to decide which chunk to send/ask, to which peer;
- A *Peer Set* data type to store neighbors in a structure describing the overlay.

The goal of the various GRAPES modules is to hide the implementation of the mechanisms introduced above, and to make them accessible through

an uniform and generic programming interface. In this way, implementation details of the various mechanisms can be easily changed without affecting the application code.

Note that although the following description of GRAPES is based on an example using all its functionalities, applications are free to use only the needed modules. For instance, a streamer based on a tree-like overlay will probably not use the Peer Sampling module; other applications can use GRAPES' Peer Sampling implementing their own chunk buffer.

5.3.1 Peer Sampling

The Peer Sampling module is used by a P2P application for joining an overlay: the application provides to the peer sampling mechanism one (or more) known peers, and can obtain a "view" containing a random sample of all the peers currently active in the system. Such a mechanism can be implemented by using a gossipping protocol like NewsCast [48] or CYCLON [100], or some other mechanism (the use of a centralized database has been proposed in some situations [65]). Currently, a simple gossipping protocol (similar to NewsCast) has been implemented, and an implementation of CYCLON will be released soon. If a specialized peer sampling mechanism is needed, it can be easily implemented in this module, and made available to all the applications using the GRAPES API. The most important functions exported by the Peer Sampling module are:

- `Init()`, to initialize the peer sampling service, and assign a local address to it
- `ParseData()`, invoked when a peer sampling message from a remote peer is received
- `AddNeighbour()`, to provide the ID of known peers; mainly used for bootstrapping

- `RemoveNeighbour()`, to remove a peer from the local view
- `GetNeighbourhood()`, which returns a random sample of the peers (a list of the known peers)
- `GrowNeighbourhood()` and `ShrinkNeighbourhood()`, to modify the size of the local view (the set of known peers)
- `GetMetadata()` and `ChangeMetadata()`, to get and modify the metadata associated to a peer.

Moreover, some *metadata* can be associated to each peer, to describe its attributes (e.g., upload bandwidth) metadata are application-specific, and are handled transparently by GRAPES.

5.3.2 Signalling and Chunk Trading

Once an application has a list of some of the peers participating to the P2P system, it can exchange signalling messages (telling which chunks it needs and/or which chunks it can provide to the other peers) and chunks with the other peers.

GRAPES provides powerful and generic signalling protocol primitives, which allow to implement a large set of different chunk trading mechanisms. Analyzing the signalling messages required by various chunk trading protocols found in the literature, it is clear that most signalling messages send a set of chunk IDs, but with different semantics. A buffermap, for example, is a set of chunk IDs encoded normally as a bitmap, just as a chunk request is usually a set with only one element.

Therefore, GRAPES provides a Chunk ID Set datatype that can be used for storing the IDs of the chunks owned (or needed, or offered, or accepted, ..., depending on the required semantics) by a peer. The signalling functionalities also provide a low-level API with a generic `encodeChunkSignaling()`

function, that takes a Chunk ID Set as a parameter, transforming it in a message to be sent on the network through the network helper. Metadata explaining the type and the semantic of the message (e.g., whether the chunks are needed or offered) and information about the chunk trading protocol (e.g. a transaction ID) can also be encoded by `encodeChunkSignaling()` and added to the message. Note that the current implementation supports different encoding schemes, e.g., Chunk ID Sets can be encoded both as bitmaps (useful for large dense sets), as well as lists (useful for sparse sets or for debugging purposes, optionally assigning priorities and other information to the various chunk instances. The different encoding schemes support different trade-offs between bandwidth usage and represented information.

Like the Chunk ID Sets, chunks can be transmitted by using appropriate `encode()` (`encodeChunk()`, for transforming a chunk or a Chunk ID Set in a message) and `decode()` functions (`decodeChunk()`, for transforming a message in a chunk).

Based on the above low-level function, an API consisting of "high-level" functions has also been built. Thanks to such a high-level API, GRAPES implements a large set of signalling protocols, which allow the composition of very different chunk trading mechanisms. The following signalling functionalities have been implemented based on `encodeChunk-Signaling()`:

- buffermap message (to inform another peer about the status of the chunk buffer)
- chunk offer (to offer a set of chunks to another peer)
- chunk accept (the response to an offer message)
- chunk request (ask for one or more chunks)
- chunk deliver (response to a request)

Various chunk trading logics can easily be obtained from the above primitives: A simple "useful" push protocol [12] uses only buffermap messages; a pull protocol [42, 109] will use request and deliver messages; and a more complex one that uses bufferstate information to send pull requests to selected peers will use all the buffermap, request, and deliver messages. More complex trading protocols, such as an offer-accept protocol are also supported: The Technical Report [3] reports examples of streamers built using GRAPES.

At the implementation level, the above messages are rather similar and therefore easily parseable. Signalling information are encapsulated in messages by a message type, by using the Chunk ID Set datatype, and some other information (encoded as *metadata*) such as a transaction ID and the maximum number of chunks to be delivered.

5.3.3 The Chunk Buffer

Chunks received by an application are generally stored in a Chunk Buffer, from which they are taken for forwarding the stream to other peers. GRAPES provides a Chunk Buffer API which enables to store the received chunks, and to get a list of the currently stored chunks. The application does not have to care about the data structure's internals, and GRAPES is responsible for ordering the chunks, removing the duplicates, discarding chunks that are too old, and other operations.

Different buffer management policies are possible:

1. the buffer discards chunks when a maximum size has been reached;
2. chunks are discarded when the difference between their playback time and the current time is too large.

Other, more advanced, policies can be designed and added to the library without affecting its interface, or the user code (for example, storing all the chunks for a larger time, which can be more useful for VoD systems).

Many parameters, like the buffer size, are configurable through the initialization call.

The functions exported by the Chunk Buffer module are:

- `cb_init()`, to initialize a chunk buffer, setting its size and some important parameters
- `cb_add_chunk()`, to insert a new chunk in the buffer
- `cb_get_chunks()`, returning an ordered list of all the chunks which are currently stored in the buffer
- `cb_get_chunk()`, returning a specified chunk from a buffer
- `cb_clear()`, to remove all the chunks from a buffer
- `cb_destroy()`, to destroy a buffer, freeing all the resources used by it

5.3.4 Scheduling

During the streaming, an application often has to select chunks to send / receive, or remote peers to contact for chunk trading. All of these decisions are performed by the *peers and chunks scheduler*. Note that the scheduling decisions to be taken depend on the chunk trading protocol that the application implements. For example, when using an epidemic streaming approach an application periodically sends a chunk to a neighbor. Hence, it needs a chunk scheduler to select the chunk to be sent and a peer scheduler to select the target peer to which the chunk will be sent. In alternative if the application is based on a *pull* protocol, it has to select a set of chunks to be requested to a neighbor, and a neighbor to which the chunks are requested: Two scheduling functions are still needed, but they work in a different way respect to the "push" schedulers. Finally, more complex protocols (such as an offer/trade protocol) can be used, but any peer has still to take scheduling decisions about the chunks to offer, and about the offers that it wants to accept.

The GRAPES scheduler provides fundamental scheduling functions that can be used in the situations described above, and are, in the authors' opinion, generic enough to be used in many other situations. Furthermore, a *scheduling framework* that can be used to implement new and more specialized schedulers is provided. The final goal is to have a scheduling API which is compatible with the one used by SSSim [13], so that schedulers can be easily moved from the simulator to real applications and vice-versa.

5.3.5 Other Modules

Other modules are currently under development and will be available in the next releases of the software. For example, a new module will contain some *topology management* algorithms such as TMan [49] that allow to build a more structured overlay based on the random view provided by the Peer Sampling module.

Another important GRAPES module which has not been fully yet allows to connect a P2P application with the `libavcodec` and `libavformat` libraries[2], to encode/decode audio and video, and to handle multimedia formats. Such a module can be used in the input and output parts of a P2P streaming application, to implement *media aware streaming* (for example, inserting an integer number of frames in each chunk, or assigning different importance to different chunks based on the presence of reference frames in them). The functionalities of this module have already been implemented, and [56] shows how to use such functionalities together with a simulator, but they have also been used in a real streamer.

[2] www.ffmpeg.org

5.4 Usage and Early Experiences

Applications based on GRAPES can use the library's public interface (exported through some C header files) to initialize the network helper and the various GRAPES modules, to send/receive messages, and to pass them to the appropriate `ParseData()` function. The typical application will

1. Initialize the various components
2. Enter a loop in which it:
   (a) Receive messages
   (b) Demultiplex them and pass them to the appropriate module
   (c) Eventually send back messages (this can be done by the module itself

Listing 5.1 shows how to do this in a single-threaded application. The `wait4data()` function, exported by the network helper, allows the application to block waiting for a message or for a timeout to fire. If a message arrives before the timeout fires, the message is received (`recv_from_peer()` and is passed to the proper `ParseData()` function, selected through a `is*()` function (in this example, only the handling of the Peer Sampling messages - identified by `isTopology()` - is shown; if the application uses more GRAPES modules, other `is*()` and `*ParseData()` functions will be invoked - in place of the "else check if the message goes to other GRAPES modules" comment).

Based on the structure described above, a simple application which builds a P2P overlay by using the Peer Sampling service has been written with about 100 lines of C code. Such a program compiles in an executable large about 10 kB, which requires less than 2 kB of data to execute.

Listing 5.1: Single-threaded usage of GRAPES.

```
struct nodeID *my_id;
my_id = net_helper_init(my_addr, my_port);
```

```
  topInit(myID, NULL, 0, "");

  while(!done) {
    new_msg = wait4data(s, &timeout, NULL);
    if (new_msg) {
      len = recv_from_peer(s, &remote, buff, BUFFSIZE);
      if (isTopology(buff)) {
        topParseData(buff, len);
      } /* else check if the message
           goes to other GRAPES modules */
      nodeid_free(remote);
    } else {
      /* Invoke Parse functions with NULL
         argument, to check for timeouts */
      topParseData(NULL, 0);
      /* Other modules' Parse() */
    }
  }
```

As previously explained, GRAPES does not force any particular application structure, so it can also be used in a multi-thread environment, as shown in Listing 5.2. Note that in this case the application is responsible of ensuring mutual exclusion on the GRAPES functionalities and data structures (by using appropriate mutexes), so GRAPES does not depend on any specific threading library. Alternative implementations of the network helper which allow to use GRAPES in event-based programs have been developed and will be integrated in the main codebase soon.

Listing 5.2: Multi-threaded usage of GRAPES.

```
void *ps_thread(void *arg)
{
  topInit(myID, NULL, 0, "");
  while(!done) {
    pthread_mutex_lock(&topology_mutex);
    topParseData(buff, len);
    pthread_mutex_unlock(&topology_mutex);
    usleep(gossiping_period);
```

```
  }
  return NULL;
}

/* Thread bodies for other GRAPES modules */
void *recv_thread(void *arg)
{
  while(!done) {
    len = recv_from_peer(s, &remote, buff, BUFFSIZE);
    if (isTopology(buff)) {
      pthread_mutex_lock(&topology_mutex);
      topParseData(buff, len);
      pthread_mutex_unlock(&topology_mutex);
    } /* else check if the message
         goes to other GRAPES modules */
    nodeid_free(remote);
  }
  return NULL;
}

int main(int argc, char *argv[])
{
  my_id = net_helper_init(my_addr, my_port);
  pthread_create(&id1, NULL, recv_thread, NULL);
  pthread_create(&id2, NULL, ps_thread, NULL);
  pthread_create(...);  /* Create threads
    for the other GRAPES modules ... */
```

To test the portability of the library, some tests have been cross-compiled for an embedded platform (an ARM-based board) and successfully tested on it. This proves that the library's dependencies are minimal, and that GRAPES-based applications can be used in resource-constrained environments.

By using the GRAPES library, a simple but functional P2P video streamer (based on epidemic streaming techniques) has been written with about 900 lines of C code. Since it is based on the GRAPES API, it is quite simple to

change the chunk or peer scheduling algorithms, the chunk buffer implementation, the peer sampling protocol, or other algorithms without large changes in the streamer's code. If compared with some previous works [66] (where more than 10000 lines of code had to be written), these results represent a considerable improvement, and enable easier experimentation with novel P2P streaming approaches. The streamer program has been developed, debugged, and tested in less than one day, and only depends on GRAPES (additional dependencies on audio/video libraries can be added to use advanced chunkisation strategies - see below); the executable size is about 26 kB (about 21 kB of code), and it needs less than 2 kB of memory for data to execute.

The generation and the playback of chunks is based on `libavcodec` and `libavformat` (as explained in Section 5.3.5, the corresponding code will be moved into GRAPES in the next releases, and these functionalities will be exported through a generic API), and can be easily modified to experiment with new media-aware chunkisation techniques (for example, using 1 GoP per chunk, or grouping frames into chunks according to their types, or using more advanced temporal scalability approaches). Moreover, it is very simple to change the video codec, or the encoding parameters, to verify which codecs/parameters are more suitable for P2P streaming applications. If the dependencies on `libavcodec` and `libavformat` are removed (by disabling support for advanced chunkisation techniques in the input and output modules), the streamer is still able to receive and forward chunks, and can be used as a *superpeer* or to improve the available upload bandwidth in a P2P system.

Thanks to the flexibility of the GRAPES API, the streamer has been modified to implement different chunk trading protocols [3], allowing to run experiments to compare such protocols through real tests on the Internet (and not through simulations).

5.5 Conclusion

In this chapter we introduced GRAPES, a toolkit for easily and rapidly developing P2P streaming applications. GRAPES provides a net helper for using the UDP protocol on POSIX systems (it has been tested on GNU/Linux, some BSDs, MacOS X, and some other POSIX compliant systems), a simple but functional implementation of the modules described in Section 5.3, and some examples and tests. An additional peer sampling algorithm (CYCLON) has already been implemented, and some additional modules are under development.

GRAPES has already been used in PeerStreamer and other experimental P2P video streaming software.

Chapter 6

Conclusion and Perspectives

The peer-to-peer communication paradigm has changed the way content is distributed over the Internet. Starting from file sharing all the way to telephony, P2P systems are fully accredited players of the Internet panorama. Streaming systems, and real time ones in particular, have been the applications posing the hardest challenges to the P2P paradigm. On the one hand, the intrinsic scaling properties of P2P seems to fit particularly well to large scale streaming. On the other hand, the strict timing constraint of smooth streaming, and in particular the limited latency required by real time applications, make solutions that are not in-network very critical.

This thesis has contributed to the field of P2P streaming in several ways, looking both into general issues like blending different approaches to swarm information diffusion and more specific topics like swarm localization without harming the application and possible extension to smaller scale (as compared to the global Internet) wireless access networks.

The first contribution has considered a new class of protocols where peers alternate push and pull diffusion schemes, called Push/Pull protocols: The push-based scheme spreads new information across the P2P network while the pull-based scheme exploits cooperative techniques to recover missed data packets from neighboring nodes. Push/Pull protocols are suitable for sup-

porting both file-based and streaming-based communications. The fundamental properties of Push/Pull protocols have been investigated, showing that the proposed asynchronous model outperforms the original cycle-based one and peers upload bandwidth influences significantly both the distribution delay and the overall system performance. Furthermore, Push/Pull protocols has been enhanced through chunks negotiation mechanism, enabling peers to establish parallel transmissions, resulting in a noteworthy reduction of the distribution delay, revealing interesting capabilities for supporting video streaming distribution; Finally, a RTT-based peer selection algorithm has been proposed to promote communications among closest neighbors. achieving lower chunks diffusion delay than the random one, showing the importance of network parameters.

The second contribution has considered the Push/Pull protocols techniques for supporting streaming multicasting in wireless mesh networks. After discussing the standard Protocol Independent Multicast and the issues that prevent PIM to work properly on WMNs, it proposed minor modifications to overcome such issues, presenting the Wireless-PIM-DM protocol; This protocol has showed interesting results in term of throughput and packet diffusion delay, being suitable for video streaming multicasting. However, multicast packets can be lost between mesh-nodes communications and while transmitting towards end-users. Therefore, the PullCast cooperative protocol has been proposed, where the video content is pushed through the multicast distribution overlay and the end-users exploit cooperative techniques to recover missed data packets from their neighboring nodes via unicast Pull transmissions. The PullCast protocol showed a significant increase in the number of peers achieving the chunks threshold (i.e. 95% of chunks), introducing a rather low chunks diffusion delay while the corresponding wireless channel load induced by the pull recovery mechanism is limited.

Finally, the last contribution has presented the GRAPES library, which provides basic functionalities for easily and rapidly developing of P2P streaming applications. Such functionalities include a collection of primitives for exchanging information among peers, performing chunks negotiation, scheduling chunks, and many other helpful primitives. GRAPES has been employed as the core library of the PeerStreamer open-source streaming application. Today, PeerStreamer is one of the most diffused open-source system for P2P streaming used by researchers and practitioners.

Papers Published

[1] L. Abeni, C. Kiraly, A. Russo, M. Biazzini, and R. Lo Cigno. Design and Implementation of a Generic Library for P2P Streaming. In *Proceedings of the 2010 ACM Workshop on Advanced Video Streaming Techniques For Peer-to-Peer Networks and Social Networking*, pages 43–48. ACM, 2010.

[2] R. Lo Cigno, A. Russo, and D. Carra. On Some Fundamental Properties of P2P Push/Pull Protocols. In *Second International Conference on Communications and Electronics (ICCE 2008)*, pages 67–73, Jun. 2008.

[3] A. Russo, M. Biazzini, C. Kiraly, L. Abeni, and R. Lo Cigno. Implementing Streamers with GRAPES: Initial Experience and Results. Technical report, TR-DISI-10-039, University of Trento, 2010. http://disi.unitn.it/locigno/preprints/TR-DISI-10-039.pdf.

[4] A. Russo and R. Lo Cigno. Push/Pull Protocols for Streaming in P2P Systems. In *IEEE INFOCOM Student Workshops 2009*, pages 1–2. IEEE, 2009.

[5] A. Russo and R. Lo Cigno. Delay-Aware Push/Pull Protocols for Live Video Streaming in P2P Systems. In *IEEE International Conference on Communications (ICC 2010)*, May 2010.

[6] A. Russo and R. Lo Cigno. Delay-Awareness in Push/Pull Streaming Protocols. Technical Report 10-012, DISI – University of Trento, Italy, Jan. 2010.

[7] A. Russo and R. Lo Cigno. PullCast: Peer-Assisted Video Multicasting for Wireless Mesh Networks. In *10th Annual Conference on Wireless On-demand Network Systems and Services (WONS'13)*, Mar. 2013.

[8] A. Russo, R. Lo Cigno, and I. Rubin. Protocol Independent Multicasting in Wireless Mesh Networks. Technical Report 12-018, DISI – University of Trento, Italy, 2012.

[9] A. Russo, R. Lo Cigno, and I. Rubin. Protocol Independent Multicast: from Wired to Wireless Networks. In *IEEE International Conference on Computing, Networking and Communications (ICNC 2013)*, Jan. 2013.

Bibliography

[10] NAPA-WINE: Network-Aware P2P-TV Application over Wise Networks. www.napa-wine.eu.

[11] IEEE Std 802.11-2007, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, Dec. 2007.

[12] L. Abeni, C. Kiraly, and R. Lo Cigno. On the Optimal Scheduling of Streaming Applications in Unstructured Meshes. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference (NETWORKING 2009)*, pages 117–130. Springer-Verlag, May 2009.

[13] L. Abeni, C. Kiraly, and R. Lo Cigno. SSSim: A Simple and Scalable Simulator for P2P Streaming Systems. In *Proceedings of IEEE International Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks (CAMAD 2009)*, Pisa, Italy, Jun. 2009.

[14] A. Adams, J. Nicholas, and W. Siadak. Protocol Independent Multicast - Dense Mode (PIM-DM), 2005. www.ietf.org/rfc/rfc3973.txt.

[15] T. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT). *ACM SIGCOMM Computer Communication Review*, 23(4):85–95, 1993.

[16] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of the 2002 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM 2002, pages 205–217, New York, NY, USA, 2002.

[17] E.W. Biersack, D. Carra, R. Lo Cigno, P. Rodriguez, and P. Felber. Overlay Architectures for File Distribution: Fundamental Performance Analysis for Homogeneous and Heterogeneous Cases. *Computer Networks*, 51(3):901–917, 2007.

[18] R. Birke, E. Leonardi, M. Mellia, A. Bakay, T. Szemethy, C. Kiraly, R. Lo Cigno, F. Mathieu, L. Muscariello, S. Niccolini, et al. Architecture of a Network-Aware P2P-TV Appli-

cation: the NAPA-WINE Approach. *IEEE Communications Magazine*, 49(6):154–163, 2011.

[19] B. Biskupski, M. Schiely, P. Felber, and R. Meier. Tree-Based Analysis of Mesh Overlays for Peer-to-Peer Streaming. In *Distributed Applications and Interoperable Systems*, pages 126–139. Springer, 2008.

[20] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg. Epidemic Live Streaming: Optimal Performance Trade-Offs. In *Proceedings of the 2008 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2008)*, pages 325–336, New York, NY, USA, 2008.

[21] C. Bradley, D. Steve, F. Bill, K. Isidor, and T. Ajit. Internet Group Management Protocol, Version 3. www.ietf.org/rfc/rfc3376.txt.

[22] D. Carra, R. Lo Cigno, and E.W. Biersack. Graph Based Analysis of Mesh Overlay Streaming Systems. *IEEE Journal on Selected Areas in Communications*, 25(9):1667–1677, 2007.

[23] D. Carra, R. Lo Cigno, and E.W. Biersack. Stochastic Graph Processes for Performance Evaluation of Content Delivery Applications in Overlay Networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(2):247–261, 2008.

[24] A. Carta, M. Mellia, M. Meo, and S. Traverso. Efficient Uplink Bandwidth Utilization in P2P-TV Streaming Systems. In *IEEE Global Telecommunications Conference (GLOBECOM 2010)*, pages 1–6, 2010.

[25] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proceedings of the 19th ACM symposium on Operating Systems principles (SOSP 2003)*, pages 298–313, New York, NY, USA, 2003.

[26] C.C. Chiang and M. Gerla. On-Demand Multicast in Mobile Wireless Networks. In *Sixth International Conference on Network Protocols*, pages 262–270, Oct. 1998.

[27] Y. Chu, S.G. Rao, S. Seshan, and H. Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, 2002.

[28] D. Ciullo, M. Mellia, M. Meo, and E. Leonardi. Understanding P2P-TV Systems Through Real Measurements. In *IEEE Global Telecommunications Conference (GLOBECOM 2008)*, pages 1–6, 2008.

[29] A.P.C. Da Silva, E. Leonardi, M. Mellia, and M. Meo. A Bandwidth-Aware Scheduling Strategy for P2P-Tv Systems. In *8th International Conference on Peer-to-Peer Computing (P2P 2008)*, pages 279–288, 2008.

[30] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. *Peer-to-Peer Systems II*, pages 33–44, 2003.

[31] S. Deering, D.L. Estrin, D. Farinacci, V. Jacobson, C.G. Liu, and L. Wei. The PIM Architecture for Wide-area Multicast Routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, Apr. 1996.

[32] P. Dhungel, X. Hei, K.W. Ross, and N. Saxena. The Pollution Attack in P2P Live Video Streaming: Measurement Results and Defenses. In *Proceedings of the 2007 Workshop on Peer-to-Peer Streaming and IP-TV*, pages 323–328, 2007.

[33] C. Diot, B.N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *Network, IEEE*, 14(1):78–88, 2000.

[34] B. Fenner, M. Handley, I. Kouvelas, and H. Holbrook. Protocol Independent Multicast - Sparse Mode (PIM-SM), 2006.

[35] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers. The Flux OSKit: A Substrate for Kernel and Language Research. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, SOSP '97, pages 38–51, New York, NY, USA, 1997. ACM.

[36] R. Fortuna, E. Leonardi, M. Mellia, M. Meo, and S. Traverso. QoE in Pull Based P2P-TV Systems: Overlay Topology Design Tradeoffs. In *IEEE International Conference on Peer-to-Peer Computing (P2P 2010)*, 2010.

[37] G. Gheorghe, R. Lo Cigno, and A. Montresor. Security and Privacy Issues in P2P Streaming Systems: A Survey. *Peer-to-Peer Networking and Applications*, 4(2):75–91, 2011.

[38] M. Handley, I. Kouvelas, T. Speakman, and L. Vicisano. Bidirectional Protocol Independent Multicast (BIDIR-PIM).

[39] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B.K. Bhargava. PROMISE: Peer-to-Peer Media Streaming Using CollectCast. In *ACM Multimedia*, pages 45–54, 2003.

[40] M. Hefeeda, A. Habib, D. Xu, B.K. Bhargava, and B. Botev. CollectCast: A Peer-to-Peer Service for Media Streaming. *Multimedia Systems*, 11(1):68–81, 2005.

[41] X. Hei, C. Liang, J. Liang, Y. Liu, and K.W. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, 9(8):1672–1687, 2007.

[42] X. Hei, Y. Liu, and K.W. Ross. IPTV Over P2P Streaming Networks: the Mesh-Pull Approach. *IEEE Communications Magazine*, 46(2):86–92, Feb. 2008.

[43] T.R. Henderson, M. Lacage, and G.F. Riley. Network Simulations with the ns-3 Simulator. *ACM SIGCOMM demonstration*, Aug. 2008.

[44] X. Hong, M. Gerla, G. Pei, and C.C. Chiang. A Group Mobility Model for Ad Hoc Wireless Networks. In *Proceedings of the 2nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 53–60, 1999.

[45] Y.C. Hu and D.B. Johnson. Design and Demonstration of Live Audio and Video Over Multihop Wireless Ad Hoc Networks. In *Proceedings of MILCOM 2002*, volume 2, pages 1211–1216, Oct. 2002.

[46] M. Izal, G. Urvoy-Keller, E.W. Biersack, P. A Felber, A. Al Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In *Passive and Active Network Measurement*, pages 1–11. Springer, 2004.

[47] K. Jain, J. Padhye, V.N. Padmanabhan, and L. Qiu. Impact of Interference on Multi-Hop Wireless Network Performance. *Wireless Networks*, 11:471–487, 2005.

[48] M. Jelasity, R. Guerraoui, A. Kermarrec, and M. Van Steen. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 79–98. Springer-Verlag N.Y., Inc., 2004.

[49] M. Jelasity, A. Montresor, and O. Babaoglu. T-Man: Gossip-Based Fast Overlay Topology Construction. *Computer Networks*, 53(13):2321–2339, 2009.

[50] X. Jin and Y.K. Kwok. Network Aware P2P Multimedia Streaming: Capacity or Locality? In *IEEE International Conference on Peer-to-Peer Computing (P2P 2011)*, pages 54–63, Sep. 2011.

[51] D. Johnson and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In T. Imielinski and H.F. Korth, editors, *Mobile Computing*, volume 353 of *The Kluwer International Series in Engineering and Computer Science*, pages 153–181. Springer US, 1996.

[52] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *Proceedings of the 11th international conference on World Wide Web*, pages 293–304, 2002.

[53] L. Junhai, Y. Danxia, X. Liu, and F. Mingyu. A Survey of Multicast Routing Protocols for Mobile Ad-Hoc Networks. *IEEE Communications Surveys Tutorials*, 11(1):78–91, 2009.

[54] J. Kang, J. Sucec, V. Kaul, Sunil Samtani, and M.A. Fecko. Robust PIM-SM Multicasting Using Anycast RP in Wireless Ad Hoc Networks. In *Proceedings of the IEEE International*

Conference on Communications (ICC 2009), ICC 2009, pages 5139–5144, Piscataway, NJ, USA, 2009.

[55] T. Karagiannis, A. Broido, N. Brownlee, K.C. Claffy, and M. Faloutsos. Is P2P Dying or Just Hiding? In *IEEE Global Telecommunications Conference (GLOBECOM 2004)*, volume 3, pages 1532–1538, 2004.

[56] C. Kiraly, R. Lo Cigno, and L. Abeni. Deadline-based Differentiation in P2P Streaming. In *IEEE Global Telecommunications Conference (GLOBECOM 2010)*, pages 1–6, 2010.

[57] T. Klingberg and R. Manfredi. The Gnutella Protocol Specification v 0.6. *Technical specification of the Protocol*, 2002.

[58] B. Krishnamurthy, C. Wills, and Y. Zhang. On the Use and Performance of Content Distribution Networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 169–182, 2001.

[59] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoğlu, D. Estrin, and M. Handley. The MASC/BGMP Architecture for Inter-domain Multicast Routing. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 93–104, 1998.

[60] S.J. Lee, W. Su, and M. Gerla. On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks. *Mobile Networks and Applications*, 7(6):441–453, Dec. 2002.

[61] S.J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia. A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols. In *19th IEEE International Conference on Computer Communications (INFOCOM 2000)*, volume 2, pages 565–574, 2000.

[62] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the Kazaa Network. In *Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP 2003)*, pages 112–120, Jun. 2003.

[63] E. Leonardi, M. Mellia, A. Horvath, L. Muscariello, S. Niccolini, and D. Rossi. Building a Cooperative P2P-TV Application over a Wise Network: The Approach of the European FP-7 strep NAPA-WINE. *IEEE Communications Magazine*, 46(4):20–22, 2008.

[64] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In *27th IEEE International Conference on Computer Communications (INFOCOM 2008)*, 2008.

[65] H.C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin. *FlightPath: Obedience vs. Choice in Cooperative Services*. Computer Science Department, University of Texas at Austin, 2008.

[66] C. Liang, Y. Guo, and Y. Liu. Is Random Scheduling Sufficient in P2P Video Streaming? In *Proceedings of the 28th IEEE International Conference on Distributed Computing Systems (ICDCS 2008)*, volume 0, pages 53–60, 2008.

[67] X. Liao, H. Jin, Y. Liu, L.M. Ni, and D. Deng. Anysee: Peer-to-Peer Live Streaming. In *25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, volume 6, pages 1–10, 2006.

[68] Z. Liu, Y. Shen, K.W. Ross, S.S. Panwar, and Y. Wang. LayerP2P: Using Layered Video Chunks in P2P Live Streaming. *IEEE Transactions on Multimedia*, 11(7):1340–1352, 2009.

[69] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer. Push-to-Pull Peer-to-Peer Live Streaming. *Distributed Computing*, pages 388–402, 2007.

[70] N. Magharei and R. Rejaie. Prime: Peer-to-Peer Receiver-drIven MEsh-Based Streaming. In *26th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1415–1423, 2007.

[71] N. Magharei, R. Rejaie, and G. Yang. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, pages 1424–1432, May 2007.

[72] A. Majumda, D.G. Sachs, I.V. Kozintsev, K. Ramchandran, and M.M. Yeung. Multicast and Unicast Real-Time Video Streaming over Wireless LANs. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(6):524–534, Jun. 2002.

[73] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized Decentralized Broadcasting Algorithms. In *26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, pages 1073–1081, 2007.

[74] A. Montresor and M. Jelasity. PeerSim: A Scalable P2P Simulator. In *IEEE 9th International Conference on Peer-to-Peer Computing (P2P 2009)*, pages 99–100, 2009.

[75] J. Moy. Multicast Routing Extensions for OSPF. *Communications of the ACM*, 37(8):61–66, 1994.

[76] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr. Chainsaw: Eliminating trees from overlay multicast. *Peer-to-Peer Systems IV*, pages 127–140, 2005.

[77] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, et al. ALMI: An Application Level Multicast Infrastructure. In *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems*, volume 3, pages 5–5, 2001.

[78] C.E. Perkins and E.M. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1999)*, pages 90–100, Feb. 1999.

[79] F. Pianese, J. Keller, and E.W. Biersack. PULSE, a Flexible P2P Live Streaming System. In *25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, pages 1–6, 2006.

[80] B. Quinn and K. Almeroth. IP Multicast Applications: Challenges and Solutions, 2001.

[81] K.R. Raghu, S.K. Bose, and M. Ma. A Queue Based Scheduling Approach for WMAN with Adaptive Augmentation. In *IEEE Wireless Communications and Networking Conference (WCNC 2008)*, pages 1374–1379, Mar. 2008.

[82] T. Rappaport. *Wireless Communications: Principles and Practice.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2002.

[83] S. Raza, G. Cheung, and C.N. Chuah. DiCoR: Distributed Cooperative Repair of Multimedia Broadcast Losses. In *5th International Conference on Broadband Communications, Networks and Systems (BROADNETS 2008)*, pages 315–322, Sep. 2008.

[84] S. Raza, D. Li, C.N. Chuah, and G. Cheung. Cooperative Peer-to-Peer Repair for Wireless Multimedia Broadcast. In *IEEE International Conference on Multimedia and Expo (ICME 2007)*, pages 1075–1078, Jul. 2007.

[85] S. Sanghavi, B. Hajek, and L. Massoulie. Gossiping With Multiple Messages. *IEEE Transactions on Information Theory*, 53(12):4640–4654, 2007.

[86] S. Saroiu, K.P. Gummadi, and S.D. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems*, 9(2):170–184, 2003.

[87] Schiely, M. and Felber, P. Tit-for-tat Revisited: Trading Bandwidth for Reliability in P2P Media Streaming. *Multiagent and Grid Systems*, 5(2):197–215, 2009.

[88] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie: A Cooperative Bulk Data Transfer Protocol. In *23rd International Conference on Computer Communications (INFOCOM 2004)*, volume 2, pages 941–951vol.2, Mar. 2004.

[89] Y.W. Sung, M. Bishop, and S. Rao. Enabling Contribution Awareness in an Overlay Broadcasting System. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 411–422, Apr. 2006.

[90] C.C. Tan and I. Rubin. Multicasting in Energy Aware Mobile Backbone Based Wireless Ad Hoc Networks. In *3rd International Conference on Broadband Communications, Networks and Systems (BROADNETS 2006)*, pages 1–10, Oct. 2006.

[91] C.C. Tan, I. Rubin, and H.J. Ju. Multicasting in Mobile Backbone Based Ad Hoc Wireless Networks. In *IEEE Wireless Communications and Networking Conference (WCNC 2006)*, volume 2, pages 703–708, Apr. 2006.

[92] D.A. Tran, K.A. Hua, and T. Do. ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming. In *22nd International Conference on Computer Communications (INFOCOM 2003)*, volume 2, pages 1283–1292vol.2, Mar. 2003.

[93] D.A. Tran, K.A. Hua, and T.T. Do. A Peer-to-Peer Architecture for Media Streaming. *IEEE Journal on Selected Areas in Communications*, 22(1):121–133, 2004.

[94] S. Traverso, L. Abeni, R. Birke, C. Kiraly, E. Leonardi, R. Lo Cigno, and M. Mellia. Experimental Comparison of Neighborhood Filtering Strategies in Unstructured P2P-TV Systems. In *Proceedings of the 12th IEEE International Conference on Peer-to-Peer Computing (P2P 2012)*, pages 13–24, Sep. 2012.

[95] R. Van Renesse. Goal-Oriented Programming, or Composition Using Events, or Threads Considered Harmful. In *Proceedings of ACM SIGOPS EW98 Support for Composing Distributed Applications*, pages 82–87, Sintra, Portugal, 1998.

[96] U. Varshney. Multicast Over Wireless Networks. *ACM Communications*, 45:31–37, Dec. 2002.

[97] N.P. Venkata and K. Sripanidkulchai. The Case for Cooperative Networking. In *Peer-to-Peer Systems*, pages 178–190. Springer, 2002.

[98] N.P. Venkata, H.J. Wang, P.A. Chou, and K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2002)*, pages 177–186, N.Y., USA, 2002.

[99] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In *Proceedings of the 14th IEEE International Conference on Network Protocols (ICNP 2006)*, pages 2–11, 2006.

[100] S. Voulgaris, D. Gavidia, and M. Van Steen. Cyclon: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[101] S. Voulgaris and A. Harwood. BooSTER: Broadcast Stream Transmission Epidemic Repair. In *IEEE Proceedings of the 12th International Conference on Peer-to-Peer Computing (P2P 2012)*, 2012.

[102] D. Waitzman, S.E. Deering, and C. Partridge. Distance Vector Multicast Routing Protocol, 1988.

[103] S. Winkler and P. Mohandas. The Evolution of Video Quality Measurement: from PSNR to Hybrid Metrics. *IEEE Transactions on Broadcasting*, 54(3):660–668, 2008.

[104] C.W. Wu and Y.C. Tay. AMRIS: A Multicast Protocol for Ad Hoc Wireless Networks. In *IEEE Military Communications Conference Proceedings (MILCOM 1999)*, volume 1, pages 25–29, 1999.

[105] S. Xie, B. Li, G.Y. Keung, and X. Zhang. Coolstreaming: Design, Theory, and Practice. *IEEE Transactions on Multimedia*, 9(8):1661–1671, Dec. 2007.

[106] J. Xiong and R.R. Choudhury. PeerCast: Improving Link Layer Multicast Through Cooperative Relaying. In *30th IEEE International Conference on Computer Communications (INFOCOM 2011)*, pages 2939–2947, Apr. 2011.

[107] W. Xiong, L. Wu, S. Ding, and C. Wu. Research on PIM-SM Multicast Routing Improvement. In *International Conference on Computer Design and Applications (ICCDA 2010)*, volume 1, pages V1–112–V1–115, Jun. 2010.

[108] S. Ye and F. Makedon. Collaboration-Aware Peer-to-Peer Media Streaming. In *Proceedings of the 12th annual ACM International Conference on Multimedia*, pages 412–415, 2004.

[109] 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, Dec. 2007.

[110] X. Zhang, J. Liu, and B. Li. On Large-Scale Peer-to-Peer Live Video Distribution: CoolStreaming and its Preliminary Experimental Results. In *Proceedings of IEEE Multimedia Signal Processing Workshop (MMSP 2005)*, 2005.

[111] 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 *24th IEEE International Conference on Computer Communications (INFOCOM 2005)*, volume 3, pages 2102–2111, 2005.

Appendix A

List of the Symbols Used

Symbol	Description
$\mathcal{K}$	Set of peers
$\|\mathcal{K}\|$	Peers set size
$\mathcal{K}_j$	The j-th Peer
$\mathcal{N}$	Neighbors Set
$\mathcal{N}_j$	Neighbors Set of j-th node
$\|\mathcal{N}\|$	Neighbors Set Size
η	Neighbor relationship
$\mathcal{C}$	Set of Chunks
$\|\mathcal{C}\|$	Number of chunks forming the stream
$\mathcal{C}_i$	i-th chunk
$\mathcal{C}(j)$	Chunks received by the j-th client
$\mathcal{C}_{bits}$	Chunk size in bit
δ_i	Deadline of the i-th chunk
P_ω	Playout window size
K_ω	Chunk received withing the playout window P_ω
$\overrightarrow{T_\mathcal{S}}$	Source's emission time
$\overrightarrow{T_\mathcal{S}}(i)$	Source's emission time for i-th chunk
$T_j(i)$	Reception time of the i-th chunk at j-th node
$\delta_j(k)$	Chunk propagation delay between the j-th node and k-th node
$\overline{\delta}_{peers}$	Average chunks delay between any pair of peers
$\overline{\delta}_\mathcal{C}(j)$	Average chunks diffusion Delay at the j-th peer
δ_p	p-th percentile propagation delay
$\overline{\Delta}$	Chunks diffusion delay averaged over all simulations
B_s	Source stream rate

B_p	Peers' bandwidth
B_{UP}	Uplink bandwidth
B_{DW}	Downlink bandwidth
α_{up}	Number of active uploads (i.e., push transmissions)
ω_{push}	Chunks window size offered in push
ρ_{dw}	Number of passive downloads (i.e., push received)
α_{dw}	Number of active downloads (i.e., pull requested)
ρ_{up}	Number of passive uploads (i.e., pull received)
ω_{pull}	Chunks window size requested in pull
δ_{min}	Minumum one-way delay
δ_{max}	Maximum one-way delay
ω	Set of chunks to either push or pull
$R_{\mathcal{N}}$	Clients distribution radius around the AP
h_t	Hello timeout
h_l	Hello loss
$\mathcal{P}_s$	Time interval between two consecutive chunk injection
$\mathcal{P}_t$	Max time to retrieve a chunk in pull
$\mathcal{I}_{\mathcal{P}}$	K_ω range for pull activation
$\mathcal{P}_m$	Max number of pull allowed per chunk
$\mathcal{P}_r$	Max number of pull reply per $\mathcal{P}_s$
BC	Broadcast data rate in Mbit/s
UC	Unicast data rate in Mbit/s
V_{eu}	Roaming nodes speed
P_{eu}	Roaming nodes pause time
Q_{thr}	Quality threshold
STA_{SSRC}	IEEE 802.11 retransmission attemps
$\mathcal{S}$	Multicast Source
$\mathcal{G}$	Multicast Group
$\mathcal{V}$	Set of Pim-Routers
$\mathcal{V}_i$	i-th Pim-Router
$\mathcal{M}$	Set of Pim-Clients
$\mathcal{M}_j$	j-th Pim-Client

Table A.1: Notation used in the thesis.