

# Scalable Streaming Multimedia Delivery using Peer-to-Peer Communication

**Poderys, Justas**

*Publication date:*
2019

*Document Version*
Publisher's PDF, also known as Version of record

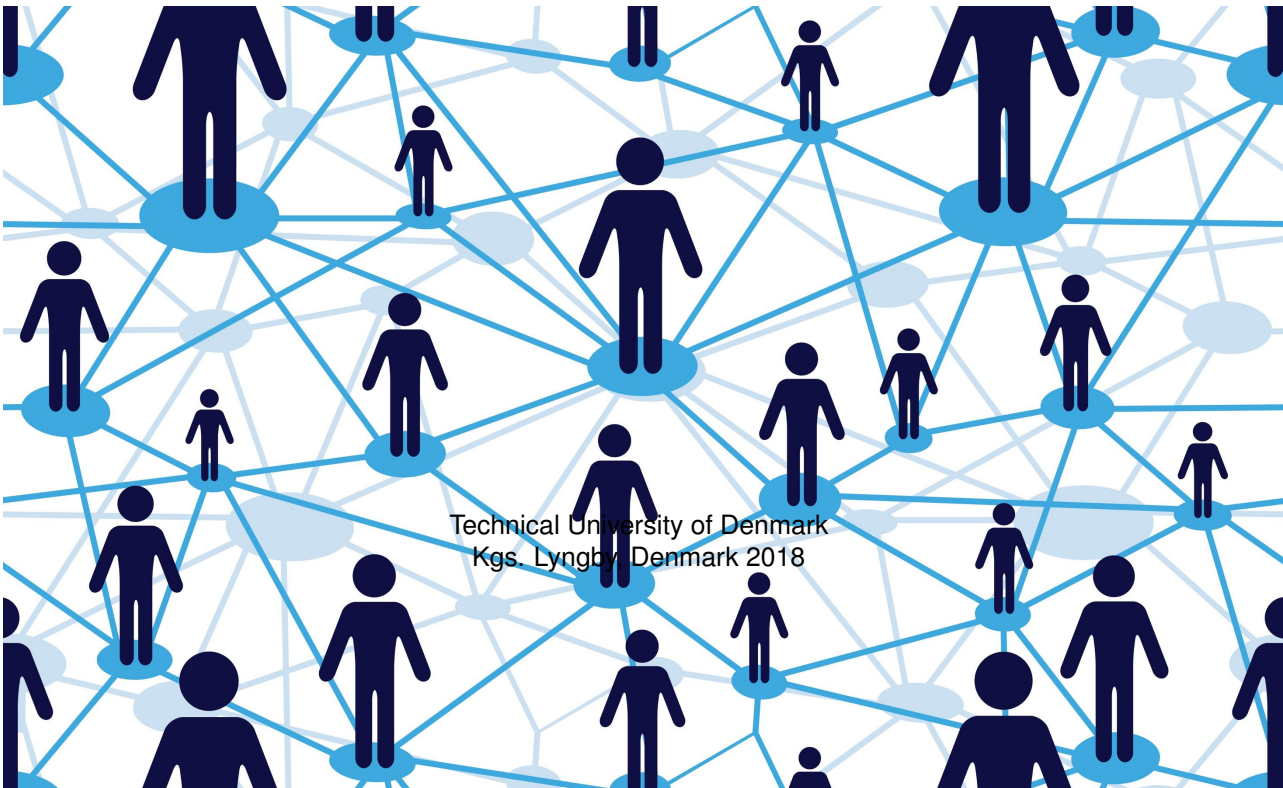
DTU

PhD Thesis

# Scalable Streaming Multimedia Delivery using Peer-to-Peer Communication

Justas Poderys

Supervised by: Jose Soler, PhD, and Prof. Lars Dittmann, PhD



Technical University of Denmark
Kgs. Lyngby, Denmark 2018

**Cover image:** ©hermione13 / 123RF Stock Photo. Print license acquired.

**Technical University of Denmark**
**Department Photonics Engineering**
Ørsteds Plads 343
2800 Kgs. Lyngby
DENMARK
Tel: (+45) 45 25 63 52
Fax: (+45) 45 93 65 81
Web: fotonik.dtu.dk
E-mail: info@fotonik.dtu.dk

# Contents

# Abstract

Peer-to-Peer (P2P) refers to a communication paradigm, where each communication peer can act as a client and as a server. In P2P communication, information is disseminated from an information source to all the clients. At the same time, as information is stored in the client's device, the clients become an information source to other clients as well.

The origins of modern P2P communication are closely linked to file distribution. The first P2P communication programs that were widely adopted were used primarily to share non-streaming data, such as music recordings, films, games, and computer programs. However, with streaming multimedia gaining a larger and larger share of the global Internet data traffic (soon envisioned to reach 80%), P2P communication is seen as a viable method to reduce the communication load on the streaming multimedia servers. By employing P2P communication along with distributed multimedia content delivery networks (CDN), high-quality multimedia data can be delivered to a larger number of users using the same amount of servers infrastructure.

This thesis investigates how P2P communication can be used to distribute streaming multimedia among multiple users. Specifically, it investigates how different algorithms, running in the Peer-to-Peer Streaming Peer Protocol (PPSPP) client, impact the start-up time and playback continuity of multimedia streams. As P2P communication is a cooperative process, this thesis investigates how requests for outstanding data should be divided among the connected peers to achieve high streaming continuity. During P2P multimedia streaming, in addition to receiving data from a multimedia server, clients also receive data from other peers. This thesis investigates how the Low Extra Delay Background Transport (LEDBAT) protocol can be tuned for use with P2P multimedia streaming while at the same time ensuring that the Internet connections of the connected peers are not congested. This thesis also presents ways to integrate P2P multimedia streaming with communication peers ranking provided by the Application Layer Traffic Optimization (ALTO) protocol.

The final two chapters of this thesis are dedicated to performance measurement of P2P multimedia streaming over different wireless communication technologies. This work investigates how P2P multimedia streaming can be performed over the Wi-Fi Peer-to-Peer connections. It also investigates methods of placing multimedia caches in the Long Term Evolution (LTE) network base-stations, along with possibilities of implementing

Peer-to-Peer communication using the same LTE infrastructure.

# Resumé

Peer-to-Peer (P2P) henviser til et kommunikations paradigme, hvor hver kommunikations deltager kan opføre sig som en klient eller en server. I P2P kommunikation bliver informationen spredt ud fra en informations kilde til alle klienter. På samme tid, fordi informationen er gemt i klientens enhed, vil klienterne blive informationskilder til andre klienter.

Oprindelsen af moderne P2P kommunikation er tæt forbundet med fildistribution. Først blev de vidt udbredte P2P kommunikationsprogrammer primært brugt til at dele ikke-streaming data, som for eksempel musik, optagelser, film, spil og computerprogrammer. Men fordi streaming multimedier optager en større og større del af den globale internetdata trafik (forudsagt til at nå 80%), bliver P2P kommunikation set som en realistisk metode til at reducere kommunikationsmængden på streaming multimedie serverne. Ved at benytte P2P kommunikation sammen med distribuerede multimedie Content Delivery Networks (CDN), kan multimediedata af høj kvalitet sendes ud til et større antal brugere, når der bruges den samme størrelse serverinfrastruktur.

Denne afhandling undersøger hvordan P2P kommunikation kan bruges til at distribuere streaming multimedier blandt et større antal brugere. Specifikt undersøges hvordan forskellige algoritmer, som kører Peer-to-Peer Streaming Peer Protocol (PPSPP) klienten, påvirker opstartstiden og playback kontinuiteten af multimedie strømme. P2P kommunikation er en samarbejdende proces, derfor undersøger denne afhandling hvordan forespørgsler fra udefrakommende data skal opdeles blandt de forbundne deltagere, for at opnå en høj streaming kontinuitet. Under en P2P multimedie streaming modtager klienterne ikke kun data fra en multimedie server, men også fra andre deltagere. Denne afhandling undersøger hvordan Low Extra Delay Background Transport (LEDBAT) protokollen kan forbedres til brug sammen med P2P multimedie streaming, mens den på samme tid ikke vil overstrømme internet tilslutningen hos de forbundne deltagere. Denne afhandling præsenterer også forskellige muligheder til at integrere P2P multimedie streaming med kommunikation deltagernes rang, givet ved Application Layer Traffic Optimization (ALTO) protokollen.

De sidste to kapitler i denne afhandling er tilegnet to ydelses målinger af P2P multimedie streaming over forskellige trådløse kommunikationsteknologier. Dette arbejde undersøger hvordan P2P multimedie streaming kan udføres over Wi-Fi P2P forbindelser. Der undersøges også metoder til at anlægge mutimedie caches i Long Term Evolution (LTE) netværks basestationer, sammen med muligheder for at implementere P2P kommu-

nikation over den samme LTE infrastruktur.

# Preface

This dissertation presents a selection of the research work conducted during my PhD studies from April 1, 2015, until March 31, 2017, under the supervision of Associate Professor Jose Soler, and Professor Lars Dittmann. It is submitted to the Department of Photonics Engineering at the Technical University of Denmark in a partial fulfillment of the requirements for the Doctor of Philosophy (PhD) degree.

This PhD project started initially as an Industrial PhD project together with company MOSAIQQ Denmark. It was converted to an academic PhD project from March 1, 2016. The work was done in the Networks Technologies and Service Platforms group at the Department of Photonics Engineering at the Technical University of Denmark (DTU), Kgs. Lyngby.

Justas Poderys
Copenhagen, 2018

# Acknowledgments

I would like to take the opportunity to thank some of the people who made this work possible. First and foremost, I would like to thank my parents. Without you, none of this would be possible. I would also like to extend my gratitude to both of my supervisors - Jose Soler and Lars Dittmann. They always had my back in all ups-and-downs of the last three years.

This thesis would not have been completed without the help of fellow PhD students. Matteo Artuso, Jahanzeb Farooq, Angelos Mimidis, Jakob Thrane, Line Hansen, Artur Pilimon, Andrea Marcano, Eder Zaballa, Cosmin Caba, and others - thank you for your support. I would also like to extend my thanks to all the great colleagues that I got to know from DTU Fotonik.

Last, but not least, I want to say thank you to my wife Laura. Your continuous love and support (especially during the last weeks of writing) made my journey to the end of this PhD project much more enjoyable. Without you I would be stuck in the academia forever.

# List of Publications

This PhD project resulted in 7 peer-reviewed publications, presented in international conferences and journals.

1. **Poderys, Justas**; Farooq, Jahanzeb; Soler, José.
   "A Novel Multimedia Streaming System for Urban Rail Environments Using Wi-Fi Peer-to-Peer Technology".
   Published in: Proceedings of 2018 IEEE 87th Vehicular Technology Conference. IEEE, 2018.

2. **Poderys, Justas**; Artuso, Matteo; Lensbøl, Claus Michael Oest ; Christiansen, Henrik Lehrmann; Soler, José.
   "Caching at the Mobile Edge: a Practical Implementation".
   Published in: IEEE Access, Vol. 6, 2018, p. 8630 - 8637.

3. **Poderys, Justas**; Sunny, Anjusha; Soler, José.
   "Implementing Resource-aware Multicast Forwarding in Software Defined Networks".
   Published in: Proceedings of the 6th World Conference on Information Systems and Technologies (WorldCist'18). Springer, 2018. (Advances in Intelligent Systems and Computing).

4. **Poderys, Justas**; Farooq, Jahanzeb; Soler, José.
   "A multimedia streaming system for urban rail environments".
   Published in: Communication Technologies for Vehicles. Springer, 2017. p. 41-53 (Lecture Notes in Computer Science, Vol. 10222).

5. **Poderys, Justas**; Soler, José.
   "Evaluating Application-Layer Traffic Optimization Cost Metrics for P2P Multimedia Streaming".
   Published in: Proceedings of 25th Telecommunications forum. IEEE, 2017.

6. **Poderys, Justas**; Soler, José.
   "Streaming Multimedia via Overlay Networks using Wi-Fi Peer-to-Peer Connections".

Published in: Proceedings of 19th IEEE International Symposium on Multimedia. IEEE, 2017.

7. **Poderys, Justas**; Soler, José.
"Using relational databases to collect and store discrete-event simulation results".
Published in: Proceedings of 30th annual European Simulation and Modelling Conference. 2016.

# List of Figures

# List of Tables

# Glossary

**3GPP** 3rd Generation Partnership Project.

**ACK** Acknowledgment.

**ALM** Application Layer Multicast.

**ALTO** Application Layer Traffic Optimization.

**AP** Access-Point.

**AS** Autonomous System.

**BBU** Baseband Unit.

**BHL** Backhaul.

**BS** Base Station.

**CIDR** Classless Inter-Domain Routing.

**COTS** Commercial Off-The-Shelf.

**C-RAN** Cloud-Radio Access Network.

**CAGR** Compounded Annual Growth Rate.

**CBTC** Communications-Based Train Control.

**CCMN** Content-Centric Mobile Network.

**CDN** Content Delivery Network.

**CN** Core Network.

**CSS** Cascading Style Sheets.

**CWND** Congestion Window.

**D2D** Device-to-Device.

**DNS** Domain Name System.

**ELM** Extreme Learning Machine.

**eNodeB** Evolved Node B.

**EPC** Evolved Packet Core.

**EPS** Evolved Packet System.

**E-UTRAN** Evolved-Universal Terrestrial Radio Access Network.

**FHL** Fronthaul.

**GTP-U** GTP User.

**HetNet** Heterogeneous Network.

**HTML** Hypertext Markup Language.

**HTTP** Hypertext Transfer Protocol.

**ICN** Information-Centric Networking.

**IETF** Internet Engineering Task Force.

**IGMP** Internet Group Management Protocol.

**IP** Internet Protocol.

**ISP** Internet Services Provider.

**JSON** JavaScript Object Notation.

**LBE** Lower-than-Best-Effort.

**LEDBAT** Low Extra Delay Background Transport.

**LI** Legal Interception.

**LTE** Long Term Evolution.

**MEC** Mobile Edge Caching.

**MME** Mobility Management Entity.

**NDO** Named Data Object.

**OS** Operating System.

**OSI** Open Systems Interconnection.

**OWD** One-Way Delay.

**P2P** Peer-to-Peer.

**PCEF** Policy and Charging Enforcement Function.

**PCI** Playback Continuity Index.

**PCRF** Policy and Charging Rules Function.

**PDCP** Packet Data Convergence Protocol.

**PDN-GW** Packet Data Network Gateway.

**PID** Provider-defined Identifier.

**PPSPP** Peer-to-Peer Streaming Peer Protocol.

**QoE** Quality of Experience.

**QoS** Quality-of-Service.

**RAN** Radio Access Network.

**REST** Representational State Transfer.

**RTT** Round-Trip Time.

**S-GW** Serving Gateway.

**SNMP** Simple Network Management Protocol.

**TCP** Transport Control Protocol.

**UDP** User Datagram Protocol.

**UE** User Equipment.

**URL** Uniform Resource Locator.

CHAPTER 1

# Introduction

Internet video delivery is by far the largest type of global consumer internet traffic. It encompass such video services as short videos delivery (i.e. YouTube), long videos delivery (i.e. Netflix), live Internet video, and Internet video for TV. It is also expected to continue growing for the foreseeable future (see Figure 1.1).

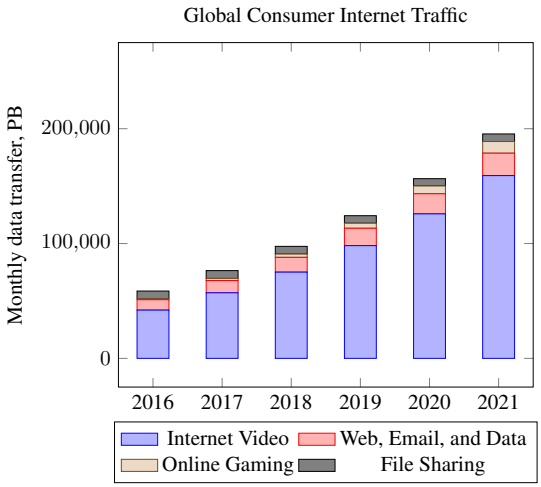
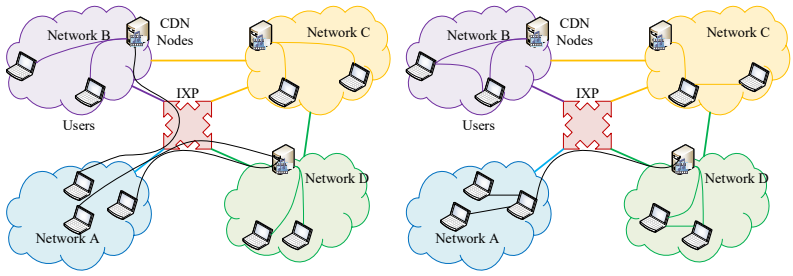



Figure 1.1: Global Consumer Internet Traffic forecast. Traffic amount by traffic type for 2016-2021. Based on data from [1].

In order to deliver high-quality streaming multimedia to its clients, multimedia content providers typically use Content Delivery Networks (CDN). CDN consists of a number of servers (CDN Nodes), distributed among different client networks. When a client requests multimedia data, for example by visiting a news website, multimedia is delivered to the user from the CDN node closest to the user, as shown in Figure 1.2a. Only if the network lacks its own CDN node, or the node does not have the required content, the client connects to the CDN node present in another network.

As the amount of Internet video traffic grows, so does the requirements for the number CDN nodes in the networks. Hence, the operators of the CDNs are forced to look for




(a) The clients in the Network A lack a local CDN node and connect to CDN nodes in other networks via the Internet Exchange Point (IXP).

(b) By using P2P communication, users can receive multimedia from other users that have already downloaded it, in addition to receiving it from CDN nodes.

Figure 1.2: Data flows between the users and streaming multimedia servers of CDN

alternative methods of multimedia data delivery that allow delivering of high-quality multimedia by utilizing less CDN nodes. One such possible method is a Peer-to-Peer (P2P) communication-based streaming servers offload.

When a user performs P2P-assisted multimedia streaming, it download multimedia data from the streaming server. Once the data is downloaded, it is made available to other users, making the user act as a streaming server as well. This brings a number of benefits. The load on the streaming servers is reduced, because users acquire parts of the data from other users, as shown in Figure 1.2b. In addition to reducing the load on the streaming servers, using the P2P communication for multimedia streaming can reduce the load on the inter-network links as well. This is achieved by performing data exchange between the users in the same network, rather than between the users from the different network.

In order to achieve the above described benefits, a number of different algorithms and protocols run in a P2P communication client. This thesis investigates how these algorithms and protocols affect the Quality-of-Service of streaming multimedia. This thesis investigates congestion control, work distribution, and peers ranking algorithms. The results presented in this thesis were obtained by performing experiments using real devices and emulated networks.

## 1.1 Thesis Organization

This thesis is organized by following the generic-to-specific structure. The main content is divided into 7 main chapters, as shown in Figure 1.3. Chapter 2 provides an introduction of the one-to-many (multicast) communication model. It also provides a survey of 55 different P2P communication protocols, and a generic description of different stages of P2P client operation. Chapter 3 is dedicated to describing a Peer-to-Peer Streaming

Peer Protocol (PPSPP) client used throughout the thesis. The same chapter is also used to describe the methods to quantify the Quality-of-Service parameters of streaming multimedia.

The work continues by describing three internal processes of the client. The methods and protocols used for congestion control are described in Chapter 4. It also describes how the parameters of Low Extra Delay Background Transport protocol were tuned for use with multimedia streaming. By using P2P communication, clients request data from other clients in addition to requesting data from the streaming multimedia server. Chapter 5 describes algorithms that govern how much data is requested from different peers. The quality of streaming multimedia depends on the quality of the connections between the peers. Chapter 6 describes how the Application Layer Traffic Optimization service can be integrated in P2P communication to rank the peers based on the different connection parameters.

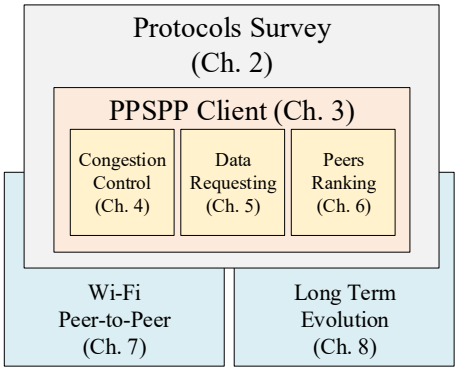



Figure 1.3: The main topics discussed in this thesis and the corresponding chapters.

The last two chapters are dedicated to two different wireless communication technologies. Chapter 7 describes an extension to the Wi-Fi standard, called Wi-Fi Peer-to-Peer, and how it can be used to form ad-hoc networks that are used to stream multimedia. Finally, Chapter 8 describes a work done on investigating the possible changes to the Long Term Evolution (LTE) networks to place streaming content caches closer to the users and enable P2P communication over the mobile base-stations.

CHAPTER 2

# One-to-Many Communication

Historically, the only group communication method supported by Internet Protocol (IP) networks was broadcast communication. A network host utilizing broadcast communication sends data to all hosts within a network's broadcast domain. However, using broadcast communication has certain limitations. First, it delivers data to all hosts within a broadcast domain without a method for opting-out from receiving such information. Second, broadcast communication spans only a single broadcast domain. Two hosts, separated by a router and hence being in two different broadcast domains, will not be able to receive broadcast information from one another.

In order to enable a more efficient one-to-many communication model, the Internet Engineering Task Force (IETF) defined methods for sending data to multiple hosts by the means of multicast communication. By implementing a multicast communication model, network hosts can indicate willingness to receive information sent to a group of hosts. Each such group is identified by a group address and hosts willing to receive information sent to the group can join and leave it at any time.

The multicast communication model solves two main problems faced by broadcast communication—it allows to efficiently use network resources and for hosts to only receive information they are interested in. However, while it is widely used in individual networks, multicast communication was never broadly adopted in the Internet [2]–[4]. Some of the issues that prevented widespread adoption were local significance of the groups, scaling issues with multicast data routing protocols and their interoperability, and lack of Quality-of-Service (QoS) management features when operating over multiple networks in different administrative domains.

Based on the end-to-end systems design approach[5], several researchers proposed to move multicast data duplication, group establishment and management function from the network layer to the application layer of the OSI model[6]. This approach is called Application Layer Multicast (ALM) and provides an alternative to network-layer multicasting. Until recently, a significant number of different application layer multicasting protocols have been proposed[7], [8]. Most of them target a narrow application area, such as file transfer or multimedia streaming. By doing so, ALM protocols move away from the "one size fits all" approach used in network layer multicasting, to the specialized "one protocol per application" used in the application layer multicasting.

Services utilizing ALM protocols are actively researched and some are already in widespread use. Peer-to-peer streaming of video content using ALM is proposed as

a solution for next generation delivery of TV programs[9]. The music distribution program "Spotify" used ALM during early, high-growth period [10]. Another well-known ALM protocol used for file distribution is BitTorrent[11], [12]. Operating systems are also starting to use ALM protocols to distribute updates to users. The latest version of Microsoft's operating system (Windows 10) is using ALM as a default method for update distribution[13]. The interest in ALM systems may continue to increase due to the standardization of device-to-device communication over Long Term Evolution (LTE) mobile networks (based on "3GPP revision 12"[14] and beyond).

Next, this chapter contains a short summary of the main features of the network layer multicasting. It is followed by a detailed description of the functioning of application layer multicast protocols, based on the survey of 56 different protocols.

## 2.1 Network-Layer Multicast Communication

Network layer multicasting[15] is a method to implement group-based communication in the IP data networks. When a computer device joins a multicast group, it receives all data sent to the group. To send data to the multicast group, computer devices address the data using the multicast group's IP address. Upon receiving data addressed to the multicast group, a first-hop router ensures that it is routed and duplicated as needed to all members of the indicated group.

Network layer multicast groups are identified by class D IP address space (range from 224.0.0.0 to 239.255.255.255)[15]. When a network device joins the multicast group, it can choose to receive all data sent to the multicast group (called any-source multicast), or only the data sent by a specific member (called source-specific multicast)[16]. The taxonomy used to indicate the network-layer multicast groups consists of a two IP addresse tuple: (S, G). The first member of a tuple (S) indicates the IP address of the source of the data and the second member (G) is a class D IP address of the multicast group. In the any-source multicast groups, the first member of the tuple can be replaced by "*", indicating that the member of the multicast group would like to receive data from all the senders.

There are two separate protocol areas in the network layer multicast network: the client access and the multicast data routing. Consider a sample network shown in panel "A" of Figure 2.1. The access area is between the end-user devices (computers and the video server) and the first-hop router, indicated by solid lines in the figure. The multicast data routing area is between the multicast enabled routers, indicated by the dashed lines in the figure. Panel "B" of the figure illustrates data flows (solid lines) from the video server to all clients in the network using network layer multicasting. Two main features of the network layer multicast are visible here: first, all data traverses each link only once, and second, routers duplicate data as it is being sent over the network.

Protocols operating in the client access area allow devices to control the membership of the multicast groups. Internet Group Management Protocol (IGMP) [17] is used

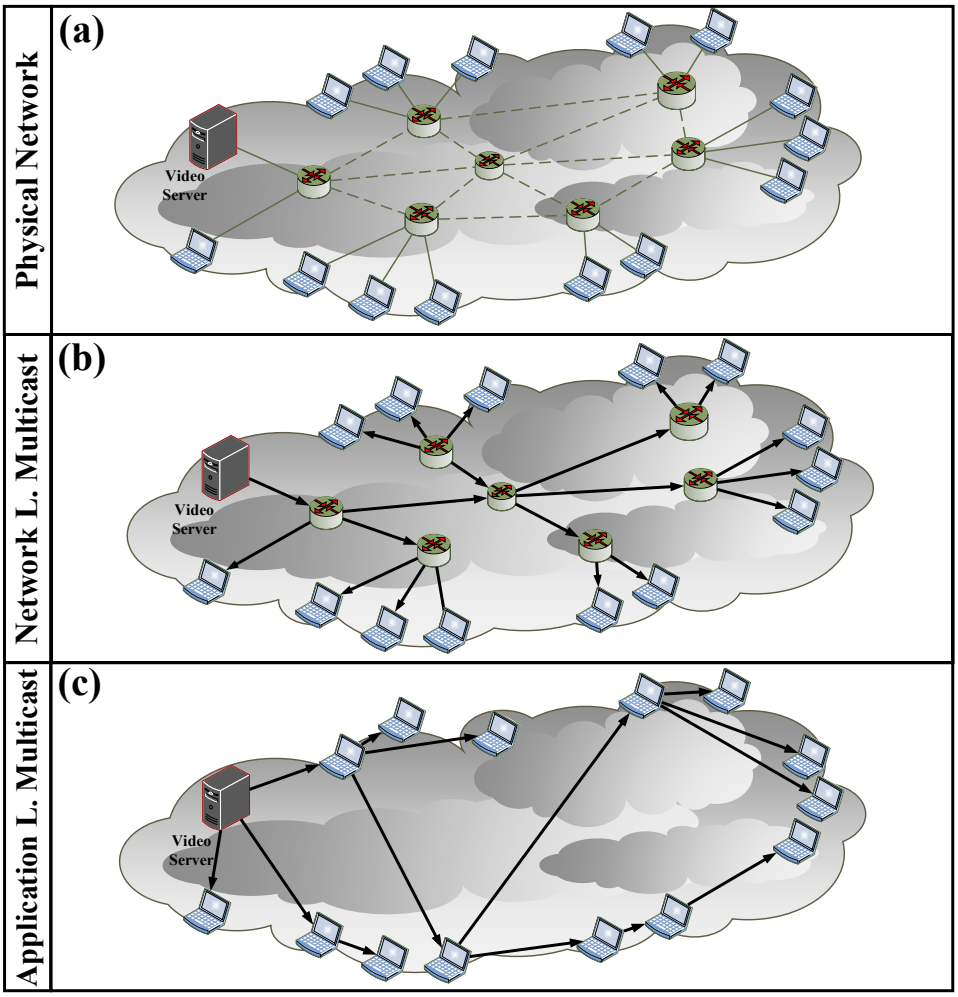



Figure 2.1: Two methods to multicast data in IP networks. A–Physical network layout; B–Data flows in network layer multicasting; C–Data flows in application layer multicasting.

to control membership of the multicast groups in the IPv4 networks and the Multicast Listener Discovery [17] (MLD) protocol is used to control membership of the multicast groups in the IPv6 networks. When a device wants to join, query or leave a multicast group, it sends an IGMP or MLD message which is processed by a first-hop router on the sending device's network. Ethernet switches can monitor IGMP or MLD messages traversing the switch to further optimize multicast data delivery in what is called IGMP snooping[18].

Protocols operating in the multicast routing area ensure that the data sent to the multicast group is delivered to all routers with a member of that group connected to them. Protocols in this area are divided into two groups. In the first group are the protocols that implement full routing protocol functionality. Among the examples of such protocols are Distance Vector Multicast Routing Protocol[19], Multicast Open Shortest Path First[20] and Multiprotocol Border Gateway Protocol[21]. In the second group are the protocols that utilize unicast routing information for multicast data delivery, and are called Protocol Independent Multicast (PIM) protocols. Examples of such protocols are PIM Dense Mode[22], PIM Sparse Mode[23], Source Specific PIM[24] and Bidirectional PIM[25].

## 2.2 Application Layer Multicasting

An alternative method to implement data multicasting in the IP networks is ALM. ALM is an umbrella term for several groups of protocols that provide data multicasting using unicast network connections. ALM protocols create direct connections between the members of the multicast group and so they follow the peer-to-peer (P2P) communication paradigm and are referred to as 3rd generation P2P protocols[26].

There are two main differences between the application layer data multicasting and network layer multicasting identified by a number of researchers[7], [8], [27], [28]. First, in application layer multicasting, data duplication is happening on the end-user device (i.e. computer, mobile phone) in contrast to the network layer multicasting, where network devices (i.e. routers) are responsible for data duplication. Second, the same data in network layer multicasting traverses any link between network layer devices only once. In application layer multicasting, the same data will traverse links between the network layer devices (i.e. between routers or routers and computers) at least once. This is because data is duplicated on the user's device, so it has to be delivered first to the user's device (1st pass over a link between the user and router) and then potentially replicated to one or more other users (2nd and further passes over a link between the user and the router). These differences are illustrated in Figure 2.1. Panels "B" and "C" show application layer data flows from the video server to client computers using network layer and application layer multicasting respectively. As shown in panel "B", data flows are split in the routers and only one copy traverses each link. Compare this with the data flows shown in panel "C", where the same data from the video server is distributed to users using application layer multicasting. Here, data is duplicated in the end-user devices and might traverse the

same link more than once, as it is being sent to and from user computers.

The logical structure that is formed by the data flows between the user devices in application layer multicasting is governed by the application layer multicast protocol. The ALM protocols are responsible for governing how users join and leave multicasting groups and how data delivery is organized. As explained later in the chapter, there are several classes of ALM protocols. While network layer multicasting protocols are standardized by the Internet Engineering Task Force (IETF), most ALM protocols are non-standardized. The only exception to this is "Peer-to-Peer Streaming Peer Protocol" standardized by the IETF in 2016. The following overview of the protocols is based on the descriptions of 55 different ALM protocols, grouped into three classes as explained in the section "Classification of ALM protocols", and given in Tables 2.1-2.3.

Table 2.1: Tree-based structured ALM protocols

| **Protocol Name** | **Year** | **Optimized for** |
|---|---|---|
| OVERCAST [29] | 2000 | Files Distribution |
| YOID [30] | 2000 | General |
| HBM [31] | 2001 | General |
| ALMI [32] | 2001 | General |
| OTBCP [33] | 2001 | General |
| AMCAST [34] | 2001 | General |
| SPREADIT [35] | 2001 | AV Streaming |
| COOPNET [36] | 2002 | AV Streaming |
| BTP [37] | 2002 | General |
| HMTP [38] | 2002 | General |
| NICE [7] | 2002 | Data Streaming |
| ZIGZAG [39] | 2003 | AV Streaming |
| OMNI [40] | 2003 | AV Streaming |
| PST [41] | 2003 | Data Streaming |
| RELAYCAST [42] | 2003 | General |
| RITA [43] | 2003 | VOD / AV Streaming |
| OSTREAM [44] | 2004 | AV Streaming |
| PROBASS [45] | 2005 | AV Streaming |
| NHAG [46] | 2005 | AV Streaming |
| CHUNKYSPREAD [47] | 2006 | VOD / AV Streaming |
| UM [48] | 2006 | General |
| ISLAND MULTICAST [49] | 2009 | General |
| TURINSTREAM [50] | 2010 | VOD / AV Streaming |
| EAGLEMACAW [51] | 2014 | VOD |

Table 2.2: Tree-based unstructured ALM protocols

| Protocol Name | Year | Optimized for |
|---|---|---|
| CAN[52] | 2001 | General Distribution |
| BAYEUX [53] | 2001 | General |
| DTP [54] | 2002 | General |
| SCRIBE [55] | 2002 | General |
| BORG [56] | 2003 | General |
| SPLITSTREAM [57] | 2003 | AV Streaming |
| THAG [58] | 2005 | Data Streaming |

Application layer multicast (ALM) protocols are classified into several distinct groups based on how the peers participating in the data distribution are organized. A survey of peer-to-peer overlay network schemes by Eng Keong et al.[7] identifies two main classes of ALM protocols: Structured and Unstructured. A survey of peer-to-peer live video streaming schemes by Zhang & Hassanein[8] classifies ALM protocols first into tree-based and mesh-based schemes. ALM protocol surveys by Hosseini et al.[27] and Tan et al.[28] group protocols into tree-first and mesh-first classes. This chapter follows the classifications proposed by Eng Keong et al. and Zhang & Hassanein. All covered ALM protocols are first grouped into the tree-based and mesh-based groups. The tree-based ALM protocols are further divided into structured and unstructured protocols. This chapter explains the core differences between the three main types of ALM protocols (tree-based unstructured, tree-based structured and mesh-based) by analyzing examples of each protocol type. Names of the ALM protocol referenced in this chapter are indicated using SMALL CAPS and references protocols listed in the Tables 2.1-2.3.

### 2.2.1 Tree-based unstructured protocols

The tree-based unstructured ALM protocols operate by organizing the nodes participating in the data distribution into a logical tree structure. Each node (except the source node) has a parent node and zero or more children nodes. The number of children nodes that each node has can be limited by the design of the ALM protocol.

The distribution of data follows the tree structure. As the data becomes available in the source node at the root of the tree, the data is duplicated to all children nodes of the source node. Upon receiving the data from the source node, all children nodes duplicate the data to the children nodes of their own. This process is repeated until the data is delivered to all the nodes in the distribution tree.

In some cases (i.e. CHUNKYSPREAD, COOPNET and SPLITSTREAM), an ALM protocol can build more than one parallel data distribution tree. Consider a system used to distribute audio/video (A/V) data encoded using Multi Description Encoding (MDC). Each description can be used to decode the A/V data, albeit with lower quality than could be possible by using more descriptions. An ALM protocol can distribute each description

Table 2.3: Mesh-based ALM protocols

| Protocol Name | Year | Optimized for |
|---|---|---|
| RMX [59] | 2000 | Data Streaming |
| GOSSAMER / SCATTERCAST [60] | 2000 | Data Streaming |
| SSOO [61] | 2002 | General |
| NARADA [19] | 2002 | General |
| ESMPCVM [62] | 2004 | AV Streaming |
| CHAINSAW [63] | 2005 | Data Streaming |
| COOLSTREAM [64] | 2005 | VOD / AV Streaming |
| GRIDMEDIA [65] | 2005 | AV Streaming |
| COOLSTREAMING+ [66] | 2007 | AV Streaming |
| PRIME [67] | 2007 | VOD / AV Streaming |
| R2 [68] | 2007 | AV Streaming |
| SUBSTREAM TRADING [69] | 2008 | AV Streaming |
| BITTORRENT [11] | 2008 | Files Distribution |
| FASTMESH [70] | 2009 | AV Streaming |
| LAYERP2P [71] | 2009 | AV Streaming |
| MTREEBONE [72] | 2010 | AV Streaming |
| SPANC [73] | 2010 | AV Streaming |
| TREECLIMBER [74] | 2010 | AV Streaming |
| OLIVES [75] | 2014 | AV Streaming |
| P2PWEBRTC [76] | 2014 | AV Streaming |
| TRANSIT [77] | 2014 | General |
| HOPES [78] | 2015 | AV Streaming |
| CYCLON [79] | 2015 | AV Streaming |
| PPSPP [80] | 2015 | VOD / AV Streaming |

over a different distribution tree, this way increasing the overall system's resiliency to failing nodes. To further illustrate the functioning of the tree-based unstructured protocol, consider OVERCAST[29] protocol. OVERCAST is a single source ALM protocol designed for bandwidth optimized data distribution. As such, it optimizes the data distribution tree for the maximum bandwidth from the nodes to the source node at the expense of end-to-end delivery latency.

Overcast is a self-organizing protocol without a centralized control. When a new node wants to join the data distribution tree, it must obtain the IP address and connect to the source node. The IP address of the source node can be entered by a user using graphical user interface or by some kind of registry service. Once the new node contacts the source

node, it starts the process of choosing the parent node as shown in Figure 2.2. The process of selecting the parent node consists of a number of rounds until the best candidate is found. At the first step (panel A), the bandwidth to the source node (indicated by S) from the new node (indicated by RN) is measured. Then, as long as bandwidth difference is no more than 10%, the bandwidth to the source node through each of the children of the source node is measured (as shown in panel B). This process of bandwidth measurement continues as long as possible while trying to put the new node as far away from the source node as possible (panel C). Eventually, the bottom of the data distribution tree will be reached, or all candidates will have insufficient bandwidth to the source node. Once the best node is found, the new node becomes the child node of that node (panel D).

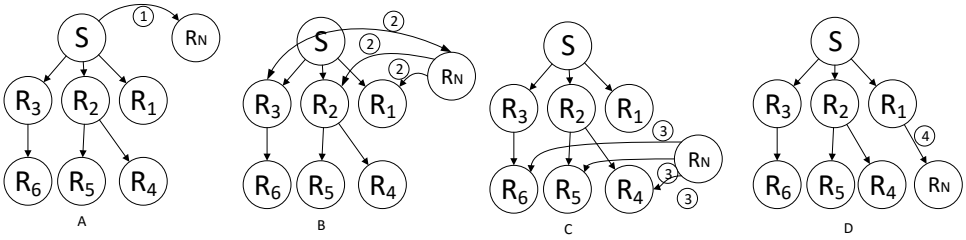



Figure 2.2: Parent node selection process in the Overcast ALM protocol. "A"—Arriving node (RN) contact the source node. "B", "C"–Arriving node goes down the data distribution tree evaluating the nodes. "D"—Arriving node positions itself as a child node of the R1 node.

The measurement of bandwidth in the Overcast protocol is done by measuring the time of 10 Kbytes data transfer from the new node to the source node. If two nodes have transfer time within 10% of each other, a traceroute hop distance is used as a tie-breaker. Nodes participating in the Overcast data distribution system periodically reevaluates its position in the data distribution tree. It does so by measuring the bandwidth to the source node to its current sibling nodes, parent and grandparent nodes. If the measured bandwidth to the root does not decrease, the node performing the measurement will try to relocate under one of its siblings. Alternatively, a node might relocate under the parent or grandparent nodes if that increases the available bandwidth to the source node. To prevent the nodes from relocating too often, a hysteresis bandwidth value can be used as a relocation step.

The last two cases to consider are the actions of the node when a node fails or leaves the system. If the parent node fails and stops relaying data, the child node contacts the grandparent nodes and tries to relocate under it to receive the data. If the grandparent node is also unresponsive, the node will continue contacting nodes up the distribution tree until it reaches the source node. If the parent node decides to leave the data distribution tree, it informs the children nodes (if any) to relocate one layer up to the grandparent

node.

### 2.2.2 Tree-based structured protocols

The tree-based structured ALM protocols tightly control the organization of the peers and distribute the data stored in the system among the peers to achieve high data queries efficiency. The placement and access of data stored in the structured system is usually based on the use of Distributed Hash Tables (DHT). In such systems, data is stored in the form of key, value, where the key is a unique random identifier used to identify a certain piece of data (value) stored in the system. In structured systems, the main function of the ALM protocol is to map the keys used to identify the data to the nodes storing this data and to route the requests to access or change the data between the participating nodes. Since the structure of the nodes formation is tightly controlled, the data distribution implicitly follows the tree form.

Next, this section presents CAN [52] protocol to illustrate the working of the tree-based structured protocol. The core of CAN design is virtual d-dimensional Cartesian space on a d-torus. All nodes participating in the multicast communication are assigned parts of this d-dimensional space. Consider an example of 2-dimensional space, as shown Figure 2.3. There are 11 nodes, each occupying a varying size of the 2-dimensional space and numbered from 1 to 11. This 2-dimentional coordinate space is used to store data having a form of key, value. All nodes know the IP addresses of the other nodes that they share a border with. The nodes self-organize themselves and implement unicast and multicast communication using the rules described next.

All data stored in the CAN space is accessed using the key value identifying the requested data. To retrieve the data having a key value of K, a deterministic hash function is applied to the key, that maps K value to the coordinates xK, yK. If the coordinates are within the area occupied by a node originating the request, it means that the data is stored on the node itself and can be accessed directly. If the point xK, yK is not in the area occupied by the node originating the request, it needs to route the data access request to the node storing the data. The request and data routing is done along the straight line path from the node requesting the data to the coordinates where the data is stored. For example, refer to Figure 2.3, where node 2 tries to access the data stored in the area occupied by node 8. To do so, it routes the data access request to its immediate neighbor along the dashed line from 2 to 8. Upon receiving the request from 2, member 5 uses the same procedure to route the data request to member 7. Member 7 then routes the data access request to member 8 using the same approach.

Using CAN routing paradigm, in a d-dimensional space divided into n equal parts, the average message routing path length is (d/4)(n1/d). At the same time, all nodes maintain 2d number of neighbors. As the number of nodes occupying d-dimensional CAN space grows, the state that each node has to maintain grows at a rate of O(n1/d) [52].

Whenever a node wants to join a CAN space, it randomly generates a point having coordinates R = xR, yR. It then routes the request to join the CAN space to the node

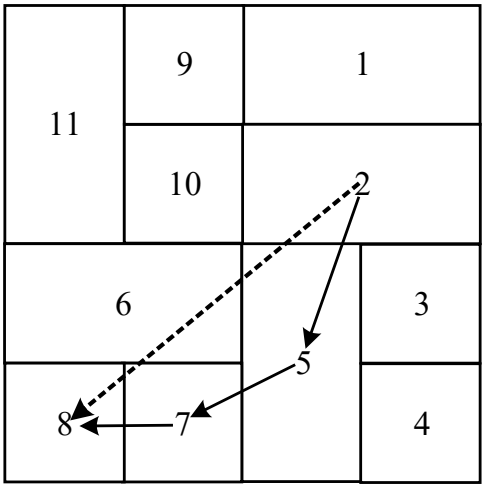



Figure 2.3: A 2-dimensional CAN space divided between 11 nodes. Numbers in the figure indicate the node's identifier. The dashed line indicates the direction of data transfer. The solid line indicates the path taken by data to travel from node 2 to node 8.

occupying the area containing the coordinate R. Upon receiving the request to join the CAN space, the node already occupying the space containing the point R splits the space in half and transfers all the data contained in the new area to the joining node. Consider the example shown in Figure 2.4. Here a new node 12 joined the CAN space by splitting the area previously occupied by node 1. Once the new node receives all the data located in its new area from the node that previously occupied it, it needs to inform the neighboring nodes about its presence. The new node does so, by sending a soft state update message to all the neighbors acquired from the previous area owner. Whenever a node wants to leave the CAN space, it follows the reverse of the join process. First, the leaving node transfers all the data located in its area to one of its neighbors. Then once the transfer is complete, the node owning a now enlarged area informs all the neighbors about the area change.

Multicasting in CAN is done by flooding a message to all members of the CAN space. In the case that not all members of a CAN space are members of a multicast group, a new instance of CAN space is created containing only the members of the multicasting group. Once this new group is established, multicast messages are flooded throughout the group. Flooding of messages in the CAN space can be implemented using several approaches. A naïve approach would be for each member of the CAN space to flood the message to all of its neighbors. While easy to implement, this approach results in nodes receiving multiple copies of the same message. In [52], the authors propose a message flooding scheme that reduces the number of duplicate messages that multicast group members receive. The proposed scheme works as follows:

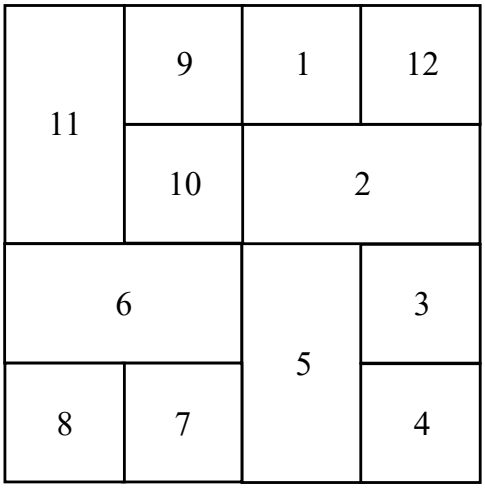



Figure 2.4: A CAN space after node 12 joins the space by splitting the area previously occupied by node 1

1. The initial node, originating the message destined to all other nodes, sends it to all of its neighbors;

2. The rest of the nodes that receive the message from their neighbor, which abuts it along dimension i, forwards the message to the neighbors with which it abuts along dimensions 1-(i-1). They also forward the message to the neighbors on the opposite side to that from which it received the message;

3. A node does not forward a message along a particular dimension if that message has already traveled more than half of the total distance possible along that dimension. This restriction prevents messages from looping back around the space.

4. All nodes cache the sequence numbers of all messages transmitted, as not to forward the messages it has previously received.

An example of messages flooding using the above rules is shown in Figure 2.5. Here a node number 2 floods a message to all other members of the CAN space as indicated by the lines representing the sent messages. From the example, it is clear, that this approach does not fully eliminate the possibility of duplicate messages. To estimate the number of duplicate messages, authors of the algorithm ran the simulations in 2-6 dimension CAN space with 16 384 nodes. "In all cases, over 97% of the nodes receive no duplicate messages and amongst those nodes that do, virtually all of them receive only a single duplicate message" [52].

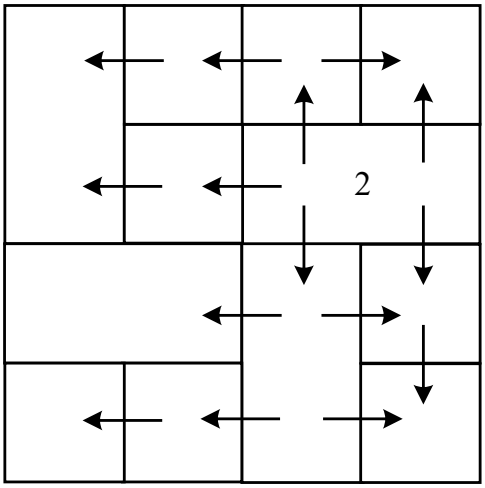



Figure 2.5: Messages flooding as used during multicasting in the CAN space.

### 2.2.3 Mesh-based protocols

Mesh-based ALM protocols form the third main class of the ALM protocols. Mesh-based protocols do not organize nodes into any specific logical form. Due to this reason mesh-based protocols are also sometimes called swarm-based protocols, because all nodes participating in the data distribution form a swarm with arbitrary interconnections. During the mesh-based protocol operation, data is "split into chunks of fixed size (in bytes or in seconds). Peers form neighboring relationships and advertise their bit-maps, which describe the chunks they have to neighbors and exchange missing chunks"[8].

As there is no strict logical structure and the peers exchange information about available data with arbitrary other peers, mesh-based protocols are more resilient to data flow interruption during peers churn (arrival and leaving). The lack of logical structure and requirement for the nodes to request the outstanding chunks made early mesh-based protocols (i.e. BITTORRENT[11]) unsuitable for live video distribution. Further work on the outstanding chunks scheduling schemes [81], [82] allowed mesh-based protocols to be used for live video streaming as well. This section continues with the analysis of Peer-to-Peer Streaming Peer Protocol (PPSPP) [80], a mesh-based ALM protocol standardized by the Internet Engineering Task Force.

In the following, messages exchanged during a PPSPP client operation are described; a graphical description with messages' structure is also given in Appendix A. The PPSPP is "specifically targeted towards streaming audio/video content and optimizes time-till-playback" [80]. It is a mesh-based protocol that supports two types of operation: when peers request missing chunks and when the missing chunks are pushed towards the peers missing them. To prevent malicious users from tampering with the distributed content,

PPSPP supports signing content with a cryptographic hash that is a root hash in a Merkle hash tree [83] built from the content being distributed. It allows the peers to verify if the downloaded data is genuine (originated by the trusted source) or injected into the system by the malicious user.

During the normal operation of PPSPP protocol shown in panel "A" of Figure 2.6, the source peer (indicated by S) is streaming data to a set of receiver peers (indicated by R). As soon as new data becomes available in the source node, the source node advertises new data to the connected peers by sending "HAVE" messages. When a receiver node learns about new data available at the connected peer, it sends a "REQUEST" message requesting this data from the peer having it (in this case the source node). When the node receives a "REQUEST" message, it replies to it with the requested data in a "DATA" message. Responding node may optionally include "INTEGRITY" message to verify the authenticity of the data contained in the "DATA" message. Whenever a receiver node receives the "DATA" message, it authenticates the content in the message (if the authentication is enforced) and advertises the availability of the new data to its connected peers by sending a "HAVE" message.

A new node joins a data PPSPP based data distribution by contacting a tracker node. The tracker node acts as a kind of registry, maintaining a list of the active node in each swarm, identified by a swarm ID. In PPSPP, each data channel or file has its own swarm ID, and one peer can participate in multiple swarms at the same time. After establishing a connection to the tracker, the new node registers with the tracker using peer-to-peer streaming protocol tracker specification[84] and receives contact information (IP address and port) of peers already in the swarm. This process is shown in panel "B" of Figure 2.6. Here, the new node (indicated by RN) contacts the tracker node (indicated by RT). The tracker responds with contact details of three nodes already in the swarm (indicated by dashed lines). The new node proceeds to connect to the nodes received from the tracker (as shown in panel "C" of Figure 2.6). To establish a connection, the new peer sends a "HANDSHAKE" message, to which the nodes already in the swarm reply with "HAVE" messages. From this point, data is exchanged between the nodes as explained above. To leave the swarm, a node sends "HANDSHAKE" message to the connected peers indicating that it is leaving the swarm and deregisters from the tracker using tracker specification.

As a node is participating in the PPSPP based data distribution, it can try to connect to other members of the swarm, in addition to the nodes received from the tracker during the initial communication. To do so, a node can either request the contact details of other nodes from the tracker node (as during the initial communication) or from already connected peers. When the second option is used, a node sends "PEX_REQ" peer request message to the connected peers. Upon receiving this message, a peer replies to the sender with a "PEX_RES" message, listing the contact details of the nodes, known to the replying peer.

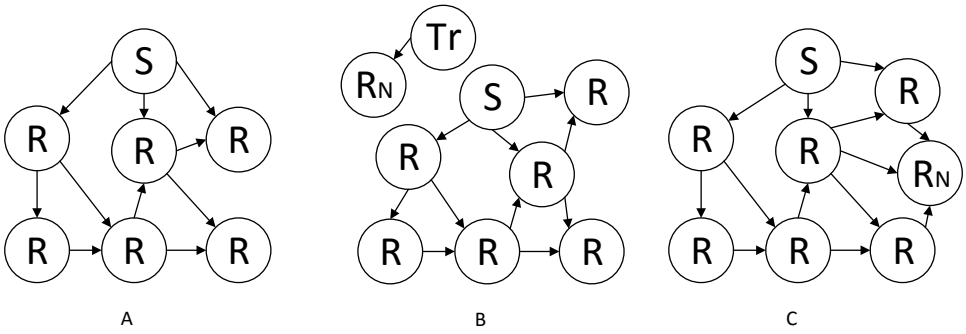



Figure 2.6: Operation stages of the Peer-to-Peer Streaming Peer Protocol. "A" – During normal operation, data is sent from the source node (S) to relay nodes (R) and between the relay nodes; "B" – An arriving (RN) node contacts the Tracker node (Tr) to learn about other nodes in the group; "C" – The arriving node starts exchanging data with other nodes in the group.

## 2.3 Application Layer Multicast Protocols Operation

This section introduces a model created to describe the operation states of application layer multicast (ALM) protocols. The model is generic and can be applied to all three kinds of ALM protocols described in this article. The model is expressed as a finite state machine (FSM) and is shown in Figure 2.7. The remainder of this section contains a discussion of each state of the FSM using examples from the surveyed ALM protocols. Names of the ALM protocol referenced in this section are indicated using SMALL CAPS and references protocols listed in Tables 2.1–2.3. All ALM protocols start operating in the "Service Discovery" state and transition through the remainder of the states during a normal operation.

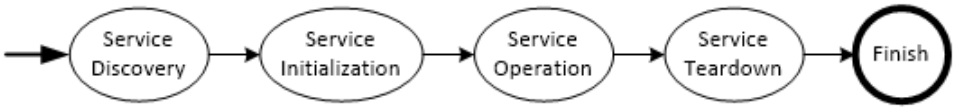



Figure 2.7: Operation states of Application Layer Multicast Protocols

The "Service Discovery" state is an initial operating state of all ALM protocols. In this state, a node running an instance of ALM protocol learns the address of a special node, called the rendezvous point (RP), which acts as the initial point of contact for all arriving nodes. Once the address of the RP is known and connection to it is established, the arriving node transitions to the "Service Initialization" state. Until then, the actual functions performed by the RP are not important for the arriving node. The RP can

be a dedicated node, or its function can be provided by a node which is the source node or a relay node at the same time. In single source systems (i.e. OVERCAST, ZIGZAG), it is common to combine functions of the RP and the source node in a single node. An alternative approach is to have the RP function provided by a dedicated node, which is neither the source node nor the relay node (i.e. in BITTORRENT, HBM, CHUNKYSPREAD, PPSPP). In systems, where the RP functionality is implemented in a dedicated node, it is called a "Tracker" node[11], [80].

There is no single way to discover the address of the RP node. Several ALM protocols (i.e. ALMI, SSOO, NICE, CHUNKYSPREAD) assume that the address of the RP node is learned using the methods outside of the scope of the ALM protocol. These methods might be the graphical user interface used by user to enter the address (i.e. in BITTORRENT, ALMI) or a link clicked in the web-site (i.e. TREECLIMBER, SPREADIT). BITTORRENT and YOID support using specially crafted uniform resource identifiers (URI) to locate the RP and a specific data distribution group. In the case of YOID, URIs have a form of "yoid://server.tld/channel:123", where "yoid" is a service identifier, "server.tld" is a hostname of the RP node and "channel:123" indicates the name of the data group and the port the client should connect to. In the case of BITTORRENT, URIs can be provided by a "Magnet-URI" framework[85]. The structure of the URI is similar to that of YOID, but the service identifier is set to "magnet" and the remaining part is set to a hash of data identifying the RP and the distributed file.

Not all ALM protocols provide a method to enter or discover the RP address. Some ALM protocols, such as the one used for Microsoft Windows 10 updates distribution[13] use the addresses preconfigured during the software development. Finally, the RP address can be distributed in an organization using a directory service[86] as proposed in ALMI protocol.

### 2.3.1 Service Initialization

Once the connection to the RP is established, a node running ALM protocol transitions to the "Service Initialization" state. In this state, the arriving node learns about other nodes participating in the data distribution, establishes connections with them and carries out any other actions required to perform protocol initialization. The arriving node transitions to the "Service Operation" state when it is ready to start data reception and distribution.

The actions performed during the service initialization are specific to ALM protocol type and ALM protocol design. In unstructured mesh-based protocols (i.e. NARADA, BITTORRENT, ESMPCVM, PPSPP, COOLSTREAMING, COOLSTREAMING+), the main objective of the arriving node is to learn the addresses of other nodes participating in the data distribution. It does so by "registering" [80] with the RP according to the protocol specification. In the case of PPSPP, the arriving nodes register with the RP by following the "Peer-to-Peer Streaming Tracker Protocol"[84]. Once the addresses of other nodes in the mesh-based protocol are known, the arriving node starts making connections to these nodes and follow the handshake procedure defined by the ALM protocol. Further learning

of participating nodes' addresses can be implemented using gossip based protocols like SCAMP [87] as used in MTREEBONE.

Service initialization in unstructured tree-based protocols is a more elaborate process due to the requirement for the arriving node to select the parent node. To find the parent node, the arriving node must build a list of candidate nodes and then find the most suitable candidate from this list. These two actions can be done either sequentially or concurrently.

In THAG and RITA protocols, the arriving node identifies its position in the network using coordinate finding and network "landmarking" systems such as GNP [88] or Vivaldi[89]. The arriving node then uses its landmark information with a small number of round-trip time (RTT) measurements to locate the logically close-by node from the already active nodes that will become a parent node.

An alternative method to network coordinates or landmark measurement is to measure network parameters to the potential parent nodes directly. In this approach, the arriving node starts measuring bandwidth (i.e. OVERCAST), RTT (i.e. PROBASS, NHAG, DTP), number of network hops (i.e. OVERCAST) or other parameter to the parent node at the root of the tree. It continues going down the data distribution tree recursively until the suitable candidate is found. Once the parent node is found, the arriving node establishes itself as the child node of the chosen parent node.

Certain protocols (i.e. YOID, OTBCP, OLIVES) use the node's IP address prefix and possibly netmask information to determine if the potential parent is logically close to the arriving node. This approach works based on the fact, that it is possible to query the public regional internet registries for the ranges of IP addresses blocks assigned to the Internet Service Providers. Then the potential parent with an address in the same IP addresses block as the arriving node is preferred to the parents with an address in other IP address blocks.

Nodes in tree-based protocols can be further grouped into groups or clusters. Each parent with its children can be considered a single cluster or nodes at the same level in the distribution tree can be considered as a cluster. Protocols like HMTP, SPREADIT, P2PWEBRTC, TREECLIMBER, ZIGZAG and NICE perform parent selection based on the cluster size. The new arriving node joins the first cluster that has the capacity to accept a new node. If no such cluster is found, a new cluster is formed either by creating a new cluster containing the new node or by splitting an already existing cluster.

Finally, in addition to recursive or distributed parent selection schemes, some protocols perform centralized parent node selection. In COOPNET, GRIDMEDIA, FASTMESH and ISLANDMULTICAST the RP node is responsible for selecting a parent node for each arriving node. To select the best parent node, the RP might require the arriving node to perform measurement of RTT to other nodes and return the results for final selection. While ALM protocol performing centralized parent node selection might appear not as scalable as protocols performing distributed selection due to the RP being a single point of failure, according to GRIDMEDIA authors "measurement results of practical application show that the workload of RP server keeps at low level with more than 15,000 simultaneous online users and thus our approach provides a considerable system capacity"

[65]. In another real-life application, authors of COOPNET observed that "at its peak, there were 18,000 nodes in the system ant the rate of node arrivals and departures were 1000 per second. (The average numbers were 10000 nodes and 180 arrivals and departures per second)" [36].

### 2.3.2 Service Operation

An instance of the ALM protocol running in a node transitions to the "Service Operation" state once the node is ready to receive data and distribute it to other nodes. The protocol instance will stay in this state until either user decides to leave the data distribution group or the data distribution stops (i.e. at the end of the live TV show). Two actions are performed by a protocol instance in this state. First, data is being received and potentially relayed to other nodes. Second, a node might try to reconfigure by making a new connection to other nodes in mesh-based protocols, or changing its position in the data distribution tree in tree-based protocols. This section will present the details on how the data is being distributed first, and then will detail how the ALM protocols perform reconfiguration.

Data distribution in tree-based ALM protocols is more straightforward than in the mesh-based protocols. In the tree-based unstructured protocols, upon receiving data from the parent node, each node makes a copy of the data for local consumption and then sends the copies of received data to all of its children nodes. An ALM protocol might change its data duplication strategy based on the priority of data being transmitted. In PST protocol, each data piece accepted by the instance of the protocol carries a priority value. Then, based on the priority value, accepted data is distributed between the nodes in the group either using Minimum Spanning Tree (MST) or Shortest Path Tree (SPT) formation.

Tree-based structured ALM protocols route data using a local routing table maintained by each node. Since the structure formed by the participating nodes is well-defined (i.e. n-dimensional space in CAN, 2-dimensional Cartesian space in DTP or circular space in PASTRY) each node needs to maintain only a small routing table with the addresses of members that are "nearby". BAYEUX uses a system similar to the longest prefix matching[90] used in the IP network instead of using geometrical space. In BAYEUX, each node has a numeric ID number and maintains local multi-level routing tables. "A node N has a neighbor map with multiple levels, where each level represents a matching suffix up to a digit position in the ID. A given level of the neighbor map contains a number of entries equal to the base of the ID, where the ith entry in the jth level is ID and location of the closest node which ends in 'i' + suffix(N, j - 1)"[53]. For example, to route a message to the node with hexadecimal ID "12ABC", a message would transition through the nodes having these IDs: "****C" - "****BC" - "**ABC" - "*2ABC" - "12ABC", where * represents any character. This way the number of entries kept by each node in its routing table is reduced.

Data in the mesh-based protocols is distributed in the group by exchanging data availability messages between the nodes. In its simplest case, each node constantly

informs connected nodes about the data available at the node. If a connected node detects, that an advertising node has a piece of data that the receiving node does not have – this missing data is requested from the advertising node. If the data being distributed is real-time, then each node might maintain a window of interest, instead of storing all the data. This approach is used in CHAINSAW, PPSPP, PRIME, COOLSTREAMING protocols. In addition to nodes requesting data from the connected nodes (pulling data), some protocols provide means for nodes to push data to the connected nodes. COOLSTREAMING+ and GRIDMEDIA support such hybrid mode of operation to faster distribute less-requested (rarer) data pieces or to minimize the messaging load between two nodes.

Data between the nodes in the data distribution group can be delivered using either UDP [91] or TCP[92] protocols. In ALM protocols, used to deliver streaming audio and video data, UDP protocol might be used together with an additional congestion control algorithm to provide equal bandwidth sharing with other application running on the same computer. Among such congestion control mechanisms used are TFRC[93] (used by GRIDMEDIA, OLIVES, COOLSTREAMING), RAP [94] (used by OLIVES) and LEDBAT [95] (used by PPSPP).

As the data transfer conditions in the Internet change over time, each node running the instance of an ALM protocol needs to adjust its position in the data distribution group. The process governing this change is shown in Figure 2.8. First, each node at periodic intervals reevaluates, whether the adjustment is needed. If adjustment is not needed, the node continues to receive and potentially relay data until another reevaluation occurs. If, during reevaluation, it is deemed that reconfiguration is needed, the instance of ALM protocol will try to reconfigure the node's position and then will transition to the wait state, restarting the whole process. Next, two sections describe the actions taken by the ALM protocols during reevaluation and reconfiguration. It is important to note that during these states, the node continues to receive the data and potentially relay it to other nodes.

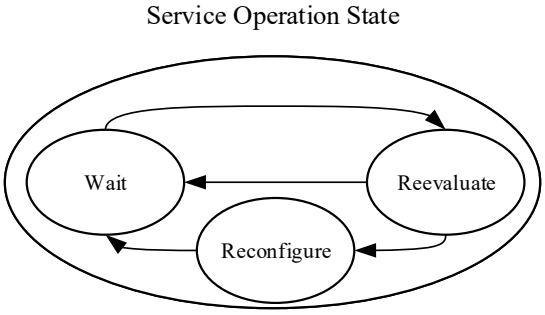



Figure 2.8: Service Operation sub-states.

### 2.3.3 Reevaluation State

A node running an instance of ALM protocol periodically enters the reevaluation state to determine if the current position in the data distribution group is optimal. The evaluation is based on the data type that is being distributed. An ALM protocol optimized for real-time streaming media distribution will aim to minimize delay and jitter (delay variation). A protocol used to distribute time insensitive data might aim to maximize the overall bandwidth or aim to exchange data "fairly"—-that is, to maintain a constant ratio of data uploaded to the neighbor node with data downloaded from the same node.

Protocols like OTBCP, NICE, BTP, HMTP use round-trip time measurement to neighbor nodes to estimate the utility of the current position in the data distribution group. YOID additionally tracks the amount of data loss observed in each node—-a parameter called "lossprint". Lossprints are periodically exchanged between the children nodes and the parent node to establish where the loss is happening. If the lossprint of the child node is similar to that of the parent, this means that the data loss is happening "above" the parent node in the data distribution tree. In such a case, a parent node is responsible for reconfiguration and not the child node.

Overcast is optimized for time insensitive data distribution, so it is using the same method as to find the parent node during joining procedure – by transferring 10 Kbytes of data to the source node through evaluated node. OTBCP also can use throughput to evaluate the utility of nodes placement.

ZIGZAG and R2 protocols aim to perform "fair" data distribution. ZIGZAG calculates the ratio of the node's degree (number of connection to other nodes) to its bandwidth capacity and to keep this value equal among the connected peers. R2 protocol operates by splitting media stream into number of a concurrent substreams. When evaluating potential peer nodes, R2 protocol instances exchange the substream maps indicating the availability of individual substreams in each node in a process called "substream exchange". If both nodes detect that each has a substream other is missing, a connection is formed in the reconfiguration state.

### 2.3.4 Reconfiguration state

An instance of ALM protocol running in a network node enters the reconfiguration state when it observes during the reevaluation state that some data transmission parameters (i.e. bandwidth, delay, ratio of exchanged traffic) are not meeting the preset threshold. Once the reconfiguration is completed, an ALM protocol instance goes back to the "wait" state (Figure 2.8) until the next reevaluation starts. The actions taken by the ALM protocol instance during the reconfiguration stance are explained next.

Mesh-based protocols do not maintain a preset logical form between the members of the data distribution group. As such, during the reconfiguration, mesh-based protocols might disconnect connections to current neighbor nodes and try to make connections to new nodes. BITTORRENT and LAYEREDP2P periodically replace the least contributing peers with new ones. The list of potential peers is obtained either from already-connected

nodes or from the RP node. NARADA follows a similar approach, but instead of comparing contribution by connected neighbors, it compares the routing tables of potential neighbor nodes. Then, based on calculated utility, it drops current connections and makes new ones.

Reconfiguration in tree-based ALM protocols is more complicated due to the strict logical structure maintained by the nodes. Reconfiguration is more straightforward in the centralized tree-based protocols (i.e. ALMI, ISLANDMULTICAST), because there is a single entity responsible for maintaining tree structure and preventing tree partitions and loops from forming. Additionally, ISLANDMULTICAST aims to build minimum-diameter degree-bounded spanning tree using an approach proposed in [34].

In tree-based ALM protocols using distributed reconfiguration (where nodes can relocate independently), each node can perform relocation in the data distribution tree as shown in Figure 2.9, adapted from the BTP protocol specification. A relocating node in the tree-based protocol can relocate anywhere, as long as it is not relocating under a node located below the relocating node in the data distribution tree, as that would create a loop in the data distribution tree.

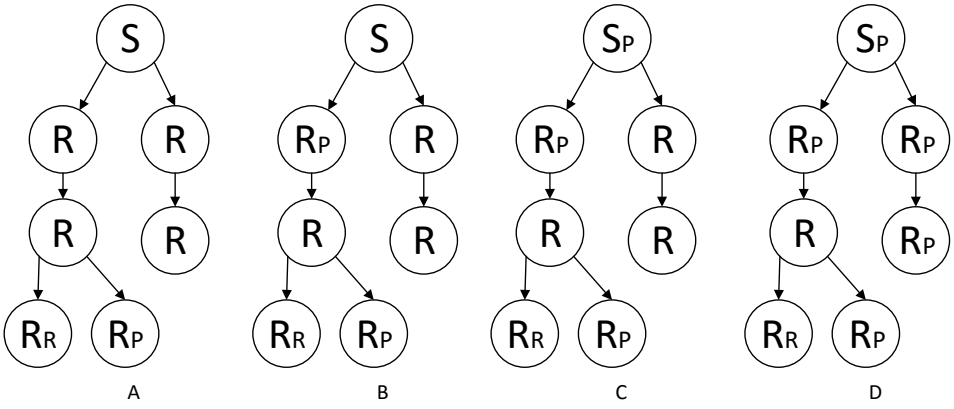



Figure 2.9: Possible relocation places (indicated with subscript P) for a relocating relay node (RR). "A" – Switch sibling, "B" – Switch one-hop. "C" – Switch two-hop. "D" – Switch any. Adapted from BTP protocol specification [37]

Tree-based ALM protocols using distributed nodes employ several approaches to prevent loops from forming during relocation. Using YOID as an example, each member maintains a rootpath—a list of nodes from the maintaining node to the source node. A node then never accepts a relocation request from a node present in a rootpath. When a node successfully relocates, it distributes its new rootpath down the distribution tree. A unique approach of using Bloom filters [96] is employed by CHUNKYSPREAD to detect and avoid loops.

Reconfiguration in ALM protocols, which maintain grouping of nodes, is usually used to keep the size of the group within certain bounds. SSO, ZIGZAG and NICE allow members to move between the groups, split groups or create new groups to maintain protocol defined group size. MTREEBONE is unique in the way that it moves nodes from the leaves of the tree towards the center (called "treebone") after a certain time that the node has been active in the group. It reflects the observation that "a node's age partially reflects its stability" [72] and more stable nodes should be in the backbone of the data distribution tree. The other objectives of tree-based ALM protocols during relocation are to form a priority based minimum spanning tree (PST) and a minimum diameter degree-bounded spanning tree (AMCAST, ISLANDMULTICAST).

### 2.3.5 Service Teardown

The final state in the lifetime of the ALM protocol instance is "Service Teardown". While in this state, the leaving node will take actions, required for gracefully leaving the data distribution. During the graceful service teardown, two major scenarios can be identified. In the first, the source node stops generating the data and all members will eventually leave the group. This happens, for example, during streaming of a movie. At the end of the movie, the source node will stop streaming data and all the nodes will leave. In the second scenario, a single node decides to leave the data distribution group, while the data distribution continues. Since the ALM protocols form cooperative data distribution systems, the departure of a single node must be handled specifically, as described in this section. One last remaining scenario is of ungraceful departure when a node crashes. This outcome is analyzed from the point of view of the nodes remaining in the data distribution group.

The teardown of data distribution service in tree-based unstructured ALM protocols requires special action from the leaving and possibly from the remaining nodes. If a node is not acting as a relay (it is a leaf node in the data distribution tree), it can leave at any time, without affecting the remaining nodes. Otherwise, the children nodes of the leaving node must relocate into the new position in the data distribution tree. The relocation can happen with the help of centralized entity (the RP node), which directs the children nodes of a leaving node to the new positions (i.e. in COOPNET, ESMPCVM, PROBASS). If the centralized approach is not used, each children node is responsible for finding its new position. It is done by re-running the join procedure (i.e. in NICE, ZIGZAG, HMTP), or by each node going up the data distribution tree and trying to relocate under each node, until a suitable new parent node is found (i.e. in OVERCAST). TURINSTREAM periodically exchanges keep-alive messages between the nodes. These messages contain "information about the peer free slots and information on its subtree" [50]. By using this information, a child node of the departing parent can quickly relocate.

The departure of the node in the tree-based structured ALM protocol is more straightforward than in tree-based unstructured protocol. Since the logical structure is clearly defined by the protocol specification, it is enough for a leaving node to inform the remain-

ing nodes (i.e. in CAN, BAYEUX, BORG). If the ALM protocols were used to distribute the data located in the nodes, a leaving node might transfer data located on the node to some other node remaining in the system (i.e. in CAN).

The leaving of the node in the mesh-based protocols is most straightforward out of the three types. As there is no strict logical structure, nodes can leave at any time, without adversely affecting data distribution. A leaving node might inform other connected nodes about its intention to leave (i.e. in PPSPP, COOLSTREAMING, BITTORRENT). A leaving node might also contact the RP node to upload the statistical information about the amount of data transferred while the node was active (i.e. in BITTORRENT).

Regardless of the ALM protocol type, nodes employ some kind of mechanism to track the liveliness of other nodes in the data distribution group. Most of the ALM protocols surveyed exchange heart-beat (HB) messages between the connected nodes (i.e. in NICE, ESMPCVM, TURINSTREAM, SPREADIT). In the mesh-based ALM protocols, periodic messages about available content can serve the function of HB messages (TFRC or BM messages in COOLSTREAMING, HAVE messages in PPSPP). If a node stops receiving HB messages from the connected node for a certain amount of time, that node is deemed to have departed and connection to it is closed by the ALM protocol. In the tree-based protocols, such ungraceful departure might cause the data distribution tree to reconfigure, as explained previously.

Several approaches can be taken to minimize the data flow interruption when the parent node is leaving. The first approach, as used in COOLSTREAMING, is for the leaving node to continue forwarding the data for some time before it leaves the data distribution system. When a user wants to leave the data distribution group (for example by closing the application), the ALM protocol can send out messages to the children nodes, requesting them to relocate immediately. Then, after some grace period, the leaving parent node will stop forwarding the data. Any children nodes not relocated by that time will experience data delivery interruptions. However, it was shown in [64] that this approach works well and the grace period does not have to be long.

The second approach to minimize the data delivery interruption is for all nodes to always maintain a backup parent node [43]. Using this approach (i.e. in RITA), each node has the main parent node and at the same time a standby parent node. In the case of the main parent node leaving, the node will be able to quickly relocate to the pre-determined backup parent node. The only requirement is for the backup parent node to not have the main parent node in its "rootpath" (list of nodes up the data distribution tree to the source node). While this approach does not eliminate the possibility that both main and standby parents will leave at the same time, it lowers the possibility of interruptions with minimal efforts from the data distribution system.

## 2.4 Summary

For a long time, network layer multicasting was the only method to implement multicasting data delivery in IP networks. Since being introduced, application layer multicasting (ALM) protocols provide an alternative method to achieve multicasting data delivery. While suffering from sub-optimal bandwidth usage compared to network layer multicasting, application layer multicasting protocols provide a number of benefits to its users. First and foremost, ALM systems work on any network infrastructure and do not require any special hardware or software support. Secondly, ALM systems do not require any configuration from the end-user, providing the real plug-and-play user experience. Thirdly, ALM systems can be tuned and tweaked for specific usage scenarios, be it real-time video distribution or file sharing, and support the reliable data transfer natively. Finally, ALM systems work well across network boundaries, not being confined to a single administrative domain.

CHAPTER 3

# Peer-to-Peer Client Structure

A software implementing peer-to-peer communication protocol is called a software client. Among the most well-known clients implementing Peer-to-Peer (P2P) communication protocols are uTorrent, implementing the BitTorrent protocol, and Microsoft Windows Updater, implementing Microsoft's proprietary P2P communication protocol. Several other popular programs that until recently had P2P communication capabilities are Spotify[10] and Blizzard Updater[97].

While there is a number of P2P communication clients available for files transfer, there are none in widespread use that are designed specifically for multimedia streaming, even though there are at least 28 different P2P multimedia streaming protocol proposals (c.f. Tables 2.1-2.3). Such the situation, when there is no widely adopted P2P streaming client, allowed to select the protocol for further experimentation based on the desirable protocol features. As the only protocol designed specifically for multimedia streaming and adopted by IETF, Peer-to-Peer Streaming Peer Protocol was selected as the protocol that will be used for experimentation.

In order for the Peer-to-Peer Streaming Peer Protocol (PPSPP) to be considered for standardization by IETF, a reference protocol implementation had to be provided. This reference implementation, written in C language, is available online under the name `libswift` project[98]. However, in order to have more freedom in design decisions, it was decided to not use the reference implementation for experiments, but rather to make an independent protocol client implementation. The client implemented for this thesis is called `PyPPSPP` (Python PPSPP), and the source-code of the client is available online (`github.com/justas-/PyPPSPP`) under the open-source license. It must be noted that during the development of `PyPPSPP` client, the reference `libswift` implementation was used to verify that the protocol is implemented correctly. It was done by having two different clients (`PyPPSPP` and `libswift`) connect to each other and transfer data.

The first part of the rest of this chapter present a functional description of all the main functional blocks of `PyPPSPP` client. Each block description consists of a brief introduction of the function performed and algorithms implemented. Most of the algorithms implemented in these block are discussed in more detail further in this thesis. The second part of this chapter discusses the specific challenges of streaming multimedia over computer networks. This chapter ends with a short overview of the methods used to quantify quality of multimedia streaming and description of quality evaluation methods used in the rest of the thesis.

## 3.1 Peer-to-Peer Streaming Software Client Structure

The structure of the PyPPSPP client software is shown in Figure 3.1. Each block in the figure is responsible for a single function of a client. Colored lines are used to indicate different data flows. Blue lines show control data exchanged between the tracker server and the client. Red lines indicate control data exchanged between the clients. Finally, green lines indicate multimedia data flows in the client and between the different clients. Each functional block in the client is implemented as a class with a well-defined interface. Such a design allows to easily replace any single class (implementing specific protocol) with a different class, without affecting the rest of the client.

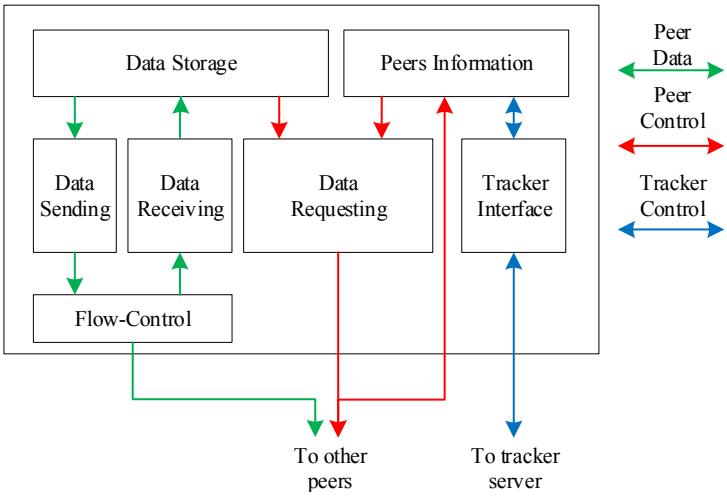



Figure 3.1: Block diagram of the PyPPSPP client

## 3.2 Tracker Interface

In a typical P2P network, the main function of the tracker server is to inform the arriving client about other clients sharing the same information in the network. Due to this reason, establishing a connection to the tracker server is one the the first actions that the P2P client performs after starting. In the PyPPSPP client, the connection between to the tracker server uses Transport Control Protocol (TCP), and an application-specific protocol in the application layer. Alternatively, some other well known tracker protocols are BitTorrent Tracker Protocol[99] and PPSPP-TP[84].

A tracker interface block in the client is used to establish the connection to the tracker and to perform data exchange between the Peers Information block and the tracker server. Upon establishing a connection, the tracker server will provide control-data containing

information about the multimedia file and list of IP addresses of other clients. The list of users must not be up-to-date or full, for as long as at least some users known to the tracker server are active. The remaining users can then be learned from the connected users using the peer-exchange mechanism of PPSPP protocol.

The connection to the tracker is not required for the client to function. However, keeping the connection open is beneficial, because the tracker server informs all connected clients when a new client arrives.

## 3.3 Peer Information

In order for the `PyPPSPP` client to function properly, it must maintain certain information about other peers. Connectivity information (IP address and port number) of other peers is learned over the tracker connection. Other information is collected directly from the peers, by making a TCP control connection to them. This connection to other peers is shown using a red line in Figure 3.1. Control connection to other peers is used to exchange initial connectivity information and to periodically inform other peers about multimedia data available locally and to request data available at the remote peers. The full handshake sequence with the handshake message contents is shown in Appendix A.

## 3.4 Data Storage

Once the multimedia data is downloaded to the client, it is stored in the data-storage block. The main function of the block is to hide the implementation details of the underlying storage method. The actual data can be stored in multiple locations. For example, it can be stored in the computer's disk-drive (if multimedia should be available after streaming), or in random access memory (RAM), in case multimedia data will be discarded after rendering. The access of data stored in the storage block is implemented by identifying each unique data piece (chunk) by its ID number. The size of the data piece can vary: in PPSPP (an in `PyPPSPP`) the default size is 1KB, but any other size can be used as well. For example, in the case of BitTorrent, each piece is between 32KB and 16MB in size. Choosing the size of the data piece is a compromise between overhead (smaller pieces have more overhead for the same underlying file size) and efficiency (smaller pieces can fit into a single IP packet without fragmentation).

Another function of the data storage block is constant discarding of already rendered data. This is done in order to reduce the amount of storage space used. For example, a client with a limited amount of storage space might not want to keep the whole stream in its memory. To limit the amount of used space, a client can maintain a discard window, indicating how many chunks it will keep in the memory after playback before discarding them. The size of a discard window (if used) is exchanged during the handshake process, and it allows other peers to track what pieces each client possesses.

## 3.5 Data Requesting

As already described in chapter 2, P2P communication can be pull-based, or push-based. In case of pull-based communication, each peer is responsible for tracking and requesting missing data chunks from the connected peers. As the PPSPP protocol is pull-based, the function of requesting information from the connected peers is implemented in the Data Requesting block.

The Data Requesting block runs an algorithm used to divide a set of all missing data chunks and send requests to the connected peers for the required parts. For the algorithm to run correctly, it must track the chunks available locally (by interfacing with data storage block), and chunks available in each of the known peers (by interfacing with the peer information block). The goal of an algorithm running in the data request block can be adjusted based on the type of information being transmitted. For offline files, the algorithm might optimize for the fastest overall delivery of the file. However, in case of multimedia streaming, the algorithm must track the playback position of multimedia stream, and optimize request, so the playback buffer is not depleted. Furthermore, a requesting algorithm might also implement a feature of aiming for fair division of work between the connected peers (to request approximately equal amount of information from each peer). Chapter 5 discusses this function in detail and presents several different algorithm implemented in `PyPPSPP` and other P2P communication software.

## 3.6 Data Sending

As described in the previous section, every peer constantly requests missing data pieces from peers advertising their availability. The process of fulfilling these requests is implemented in the data sending block. During its operation, the data sending block receives data requests and if data is available, sends the requested chunks to the requesting peer.

It is important to note that not all requests will be fulfilled. During normal peer operation (ignoring erroneous or malicious requests), two situations can occur when the received request will not be fulfilled. First, if a client is configured to maintain a discard window (described in the section on data storage), it is possible that some data chunks will be discarded before the request for them arrives. In such a case, the request for no-longer existing chunks is ignored. Second, if a data sending algorithm is configured to maintain a specific ratio between sent and received data chunks, then requests for data can be ignored in order to achieve the required data sending and receiving ratio. If a required ratio is not met, a peer trying to maintain a required ratio will send a choke message to the remote peer. It is used to informing the remote peer that all further requests for data will be ignored until an unchoke message is sent.

The processing of data requests can follow several queuing disciplines. The most straightforward, and least computationally-intensive method is to process the requests in

*first-in-first-out* (FIFO) manner. This approach is used in all the experiments described in this work. However, several alternative methods can also be implemented. One feasible alternative could include a priority queue. Requests coming from peers that upload more data, could be served first as a reward. Another feasible alternative could consider a sorted queue. This way, requests for "older" data (having lower sequence numbers) would be processed first. Such a processing discipline would improve the playback continuity of other peers at the expense of local peers computational resources, used to constantly maintain a sorted queue.

In the data-plane, the flow-control block interfaces with the data storage and flow control blocks. Once the required data pieces are retrieved from the data storage block, they are delivered to the flow-control block for sending. Depending on the method of flow control (described next), this interface to flow-control block can be internal or external.

## 3.7 Flow Control

Conventional applications typically use the User Datagram Protocol (UDP) or TCP as a transport protocol for communication. While UDP is a minimalistic, message-oriented protocol providing optional data integrity control, TCP provides reliable, ordered stream-based data transmission. However, as summarized here and later explained in detail in Chapter 4, P2P communication has unique requirements. P2P communication requires error-free data delivery, but it can deal with erroneous packets in the application layer. As data shared in P2P systems are typically split into multiple pieces, ordered delivery in not required or can also be achieved in the application, instead of transport, layer. Finally, early P2P-based file exchange application were aggressive bandwidth users[100] as they used UDP to send and receive data without any congestion control. To reduce the impact of a P2P application on network bandwidth, a number of lower-than-best-effort congestion control protocols were proposed[100]. The proposed protocols ensured that P2P application would use all the bandwidth available to the user, but would reduce the sending rate, as soon as another application started using the user's connection concurrently. The `PyPPSPP` client uses Low Extra Delay Background Transport (LEDBAT)[95] as a congestion control protocol. Chapter 4 describes specific challenges of using LEDBAT for congestion control when streaming multimedia and proposes solutions to overcome these challenges.

## 3.8 Data Receiving

The data receiving block is the simplest of all covered blocks in terms of function performed. During the operation of a client, and based on configuration parameters, the data receiving block performs two functions: data integrity checks and generation and sending of acknowledgement packets.

Data integrity checks performed by the block are used to ensure that the received data was received error-free. Errors in the data can be introduced either by using an unreliable data transmission protocol (for example UDP), or by malicious users. In the case of the PPSPP protocol used in this work, the validity of data is checked by the help of data hashing. As each shared multimedia stream is identified by its Merkle Tree Hash[83], even a single changed bit will result in failure to compute a hash. Since PPSPP support peak hashes functionality, a client can verify the validity of as little as a single chunk of data, without waiting for the full file to be downloaded.

The second function performed by the data receiving block is generation of acknowledgement (ACK) packets. When the PPSPP client is used together with unreliable UDP protocol, ACK packets are used to indicate a successful reception of a data chunk from a sending peer. Furthermore, when the PPSPP client is used with the LEDBAT congestion control algorithm, ACK packets are extended with a ACKed packet reception timestamp. This timestamp is then used by the sending peer to monitor the queuing delay. The algorithm used to estimate the queuing delay and infer the presence (or lack thereof) of a congestion is described in detail in Chapter 4.

To reduce the load on the network, in terms of both packets-per-second and bandwidth, the sending of ACK messages can be delayed. The data receiving block can implement delayed ACK functions. When enabled, a client acknowledges the reception of more than one data chunk with a single ACK packet. While this function is implemented in the PPSPP client used in this work, the delayed ACK functionality was not used in the experiments. This was done mainly to have more precise measurements of a queuing delay. Furthermore, as all experiments were performed using a networks emulator, network bandwidth and sent packets count were not the limiting factors.

## 3.9 Multimedia Streaming

The first P2P clients to be widely adopted (such as Napster, eMule and Kazaa) were designed to distribute non-streaming data files. Among the most popular types of files distributed using early P2P software were music files, software packages and movies. Distributing such data was easy, because the shared file could be split into multiple small pieces, and these pieces would be downloaded or uploaded between the peers as requested.

However, distributing streaming data using P2P communication is much more difficult. This is mainly due to the temporal characteristics of streaming data. These characteristics can be summed up using the following requirements:

1. once the playback of streaming data has started, multimedia data must be delivered before the playback can be performed;

2. in order to provide high Quality-of-Experience for the viewer, streaming multimedia should not be interrupted;

3. if a live recording is being streamed, data can not be delivered to the viewers before it has been produced.

The remaining part of this section will focus on the characteristics of multimedia data that are relevant to the process of multimedia streaming and in the scope of the three requirements given above.

### 3.9.1 Multimedia Data and its Encoding

As name suggests, multimedia files typically (but not always) consist of several different medias–sound, video and text. In order to reduce the bandwidth of each media, it is typically compressed before transmission. A device or software tool that is used to compress multimedia data is typically referred to as a *codec*. Some of the well-known audio codecs are MPEG2 Layer 3 (MP3), Vorbis and FLAC. Similarly, in order to reduce the bandwidth of video data, it can be compressed using the codec such as MPEG4, H.264 and VP8.

A multimedia stream compressed using a codec is divided into a number of data frames. The amount of frames depends on the length of the multimedia stream and the frame-rate – the amount of data frames produced from each second of multimedia stream. Typical and most widely used frame-rates are 24, 25 and 30 frames per second (fps); however, much higher frame-rates (up to thousands of frames per second) are possible and used in scientific applications or action movies.

Depending on the type of codec, the produced frames can be independent (using inter-frame encoding), or linked (using intra-frame encoding). Codec producing inter-frame encoded streams (such as Motion JPEG), encodes each frame as an independent picture. In order to decode such a stream, the decoder decodes and displays (renders) each frame independently, without maintaining any state data between the frames. This makes inter-frame encoding algorithms easy to implement in both encoder and decoder.

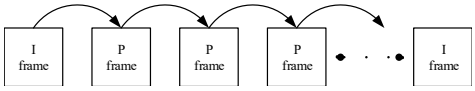



Figure 3.2: A sequence of I and P frames.

The second group of codecs exploit the similarity between the frames in a process called intra-frame encoding. An intra-frame encoded stream will produce at least two types of frames, with some elaborate encoding schemes producing as many as four different frame types. For example, two main frame types used by the H.264/MPEG-4 codec are I-frames (intra-coded picture) and P-frames (predicted picture). An I-frame typically encodes a full picture, just like in the inter-frame codec. However, by exploiting the fact that each frame is very similar to the previous one, P-frames will encode only a

difference from the previous frame. This allows to significantly reduce the bandwidth of the video stream while maintaining high picture quality.

In order to see the difference between the size of I and P-frames, consider Figure 3.3. The figure plots a size of each frame in a VP8-encoded video clip. I-frames are plotted in orange and in the whole video clip there are 5 I frames, with an average size of 29412 Bytes. In addition, there are 505 P-frames (plotted in blue), with an average size of 6035 Bytes.

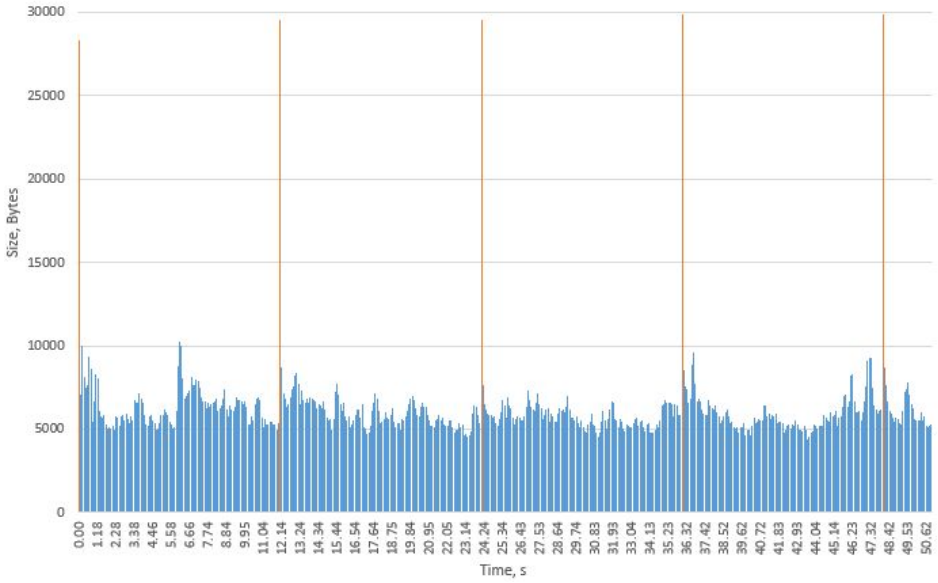



Figure 3.3: Frame sizes of the VP8 test video used in this thesis.

Such multimedia data sources have several unique challenges when being streamed using P2P communication. First, the bandwidth required to deliver multimedia stream varies due to the large differences in the sizes of data frames. In the given example, the I-frames are on average 4.9 times larger than the P-frames. High variability in the required bandwidth means that the congestion control algorithms have to react quickly to the changing bandwidth demands. As typical P2P communication users are connected using "last-mile" technologies (such as DLS, FFTx, LTE, etc.) that are more prone to congestions[101], the need for fast acting congestion controls are further increased.

Second, because decoding the P-frames depends on the preceding I-frame being available, the playback of the multimedia stream will depend on timely delivery of the I-frames. Taking the video clip shown in Figure 3.3 as an example, none of the P-frames during the first 12 seconds of the video will be decodable, without first having received an

initial P-frame.

In conventional streaming from a single streaming server, this does not pose a problem, because ordered and error free data delivery can be performed with the help of transport layer protocols, such as TCP. In P2P communication, requests for data are spread between a number of (potentially unreliable) distributed users. Should a user that received a request for data pieces containing a P-frame become offline, the playback will halt until the fact that user is gone is detected and another user sends the required data pieces.

All multimedia streaming experiments described in this thesis used VP-9 encoded multimedia stream, consisting of I- and P-frames. However, it is important to discuss several alternative encoding schemes – mainly layered coding and multiple description coding.

Multimedia streams encoded using layered coding[102] generate data divided into the base layer and a number of enhancement layers. Data in the base layer is required to render the multimedia stream. Data in enhancement layers are optional; however, applying each enhancement layer improves the quality of the rendered multimedia. Similarly to the layered coding, multiple description coding also encodes multimedia data to multiple data streams. However, in contrast to layered coding, multiple description coding does not produce a base layer. Any single data stream produced in multiple description coding can be used to render multimedia stream. Increasing the number of data streams in the rendering process increases the quality of multimedia.

While both layered and multiple description coding could be used with P2P multimedia streaming, experiments were done with single description coding mainly due to two reasons. First, using single description coding allows the system to be simplified – it is enough to optimize data delivery of a single stream instead of tracking data delivery of multiple layers or descriptors. The second reason is related to the quality-of-service parameters used to evaluate multimedia streaming performance in this thesis. As will be described later in this chapter, this thesis uses objective parameters to evaluate P2P multimedia streaming performance. Using layered coding or multi-description coding would introduce a subjective measurement of the picture quality of multimedia streaming, increasing the complexity of the work and reducing the possibility of making protocol-level adjustments in response to the quality of the multimedia stream.

### 3.9.2 Multimedia Data Buffering

In order to minimize the impact of data availability on the continuity of multimedia playback, multimedia data is typically buffered before being rendered. The following section discusses several important data buffering concepts that are later used to evaluate the Quality-of-Service of streaming multimedia.

Multimedia playback buffer can be modeled as a queue data structure, as shown in Figure 3.4. Panel A in the figure shows a playback buffer before the playback has started. In the given example, the buffer contains up to $N$ data chunks. New data chunks are being

filled from the bottom towards the top. Once the buffer is filled to the playback start start threshold (9 chunks in the given example), the playback starts.

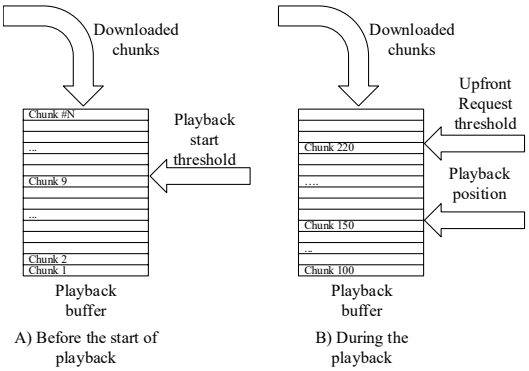



Figure 3.4: Buffering concepts

Panel B of Figure 3.4 shows the same buffer during the multimedia playback. During the playback, data pieces are removed from the bottom of the buffer. Throughout the rendering process, two indicators are tracked in the buffer. The playback position indicates which data piece is currently used for the playback. The upfront request threshold indicates the level of the buffer that will be maintained by requesting data chunks from other peers. In the given example, the upfront request threshold is 70 data chunks, meaning that the client will try to request up to th 220th chunk. An example of upfront request threshold can be seen in a screen-shot of the YouTube player shown in Figure 3.5.


Figure 3.5: Example of buffering in YouTube.

Here, the red area in the playback indicator shows the current playback position (40 seconds). The white area shows the amount of data already downloaded, and finally the grey area indicates part of the time-line, which is yet to be downloaded.

The choice of upfront request threshold is a compromise between the need to have continuous playback and effective usage of bandwidth. The higher threshold value will allow to mask the interruption in the data downloading process. However, having too much data downloaded when the user decides not to continue the playback means that the bandwidth used to download that data was wasted.

## 3.10 Multimedia Streaming Quality-of-Service Parameters

This thesis deals with the optimization of multimedia streaming using P2P communication technologies. In order to evaluate the improvement of the quality of the multimedia streaming service, there is a need to quantify such improvements. The following section will present possible methods of multimedia quality evaluation and present two parameters used in the remainder of the thesis.

There are basically two categories of multimedia quality assessment methods[103]: subjective methods that involve people as observers and evaluators, and objective methods that use mathematical models to estimate the quality of multimedia.

In subjective quality assesment methods, groups of people are given the task of evaluating the perceived quality of multimedia. Such evaluation is typically carried out in the controlled environment using standardized testing methods. Among the most well-known subjective evaluation methods is International Telecommunication Union's (ITU) Mean Opinion Score (MOS) test[104]. During this test, a group of observers evaluate rendered multimedia using the 5-point absolute category rating (ACR) scale. In the ACR scale, the perceived multimedia is given a quality score from 1 (Bad) to 5 (Excellent). MOS score is obtained by averaging individual ACR ratings of observers.

While subjective quality testing provides the most accurate results in terms of quality perceived by humans, these test are impractical to carry out and repeat. In order to provide simplified and repeatable testing, methods from the objective testing group can be used. These methods are based on mathematical models (such as Root Mean Square Error or Pearson Correlation) to evaluate the quality of multimedia. Per ITU recommendations, objective techniques are grouped into 5 groups[103], [105]:

1. media-layer models, only utilizing audio and video samples to estimate quality;
2. parametric packet-layer models, using only packet-layer information (headers) to estimate quality;
3. parametric planning models, employing encoding and network parameters to estimate quality;
4. bitstream-layer models, predicting quality from encoded bitstream and packet-layer data;
5. hybrid models, predicting quality information using one or more of the above-mentioned methods;

In order to properly select a method used to evaluate the quality of streaming multimedia, it is important to consider how multimedia will be delivered, and what is the multimedia playback model. In this thesis, multimedia is always delivered using a reliable delivery method. As such, all multimedia data available at the multimedia source will be delivered to the multimedia consumer. Furthermore, since reliable delivery methods

are used, multimedia is not altered during the transmission. Due to the above-mentioned reasons, quality estimation methods using pixel data cannot be used.

In order to estimate the quality of the multimedia streaming service, where multimedia data is not altered, several different quality models are proposed[106]. Instead of measuring difference in picture quality, these models measure the multimedia delivery parameters influencing perceived quality-of-service. Some of the parameters measured by these models ([107]–[110]) are:

1. multimedia stalling length;
2. multimedia stalling frequency;
3. number of multimedia stalls;
4. average stalling length per segment;
5. bitcount of the stalling segment;
6. initial buffering length;

In order to evaluate the performance of multimedia streaming, the experimental results presented in this thesis use multimedia stalling length and initial buffering lengths as streaming quality parameters. In accordance with other works in the field ([65], [66], [81]), the value of stalling length is expressed as a ratio, called the Playback Continuity Index (PCI). Using the PCI expressed as a ratio instead of an absolute value allows direct comparison between multimedia clips of different lengths.

The value of PCI is calculated as a ratio between the number of frames rendered in time and the number of all attempts to render the frames. For example, consider a video stream of 100 frames encoded with 25 fps frame-rate. The rendering engine operates at a fixed speed, trying to render a frame every 0.04 seconds. Every time it successfully renders a frame, it increases the number of frames rendered and the number of render attempts. Every time there is not enough data to render a frame, only the number of attempts is increased. Taking the example stream, if the rendering engine successfully rendered 80 frames and missed 20 frames (of total 100 attempts), the PCI values would be 0.8. If all frames were rendered on time, the PCI value would be 1.

This calculation assumes that the decoder does not skip the unavailable frames, but rather waits for them to be available, following the implementation of the multimedia streaming sites, such as YouTube and others.

The calculation of the initial buffering time is more straightforward. In this work, the initial buffering time is calculated as a time period from the moment the client software is initialized and connected to the tracker server, and the time the first frame is rendered.

## 3.11 Summary

As with all network communication protocols, a software implementing a protocol is required in order to use it. This chapter presents a structure and a functional description of a P2P communication client implementing PPSPP protocol. This clients was used to perform all P2P communication experiments presented in this thesis.

As with most programming tasks, choosing a protocol and selecting a client required numerous compromises. The selection of P2P streaming protocol was made easier by the fact, that PPSPP is the only P2P multimedia streaming protocol standardized by the IETF. A choice to implement a client rather than use a reference implementation provided by the PPSPP protocol designers was based on the premise that it is easier to extend a well-understood client, rather than a client implemented by other people.

This section presents the main functional blocks of the client, along with the descriptions of tasks performed by each block. Dividing the client into function-specific blocks allowed to perform single-function-specific experiments without impacting the other function. For example, such architecture of the client allows to test different congestion control schemes (see Chapter 4), data-requesting algorithms (see Chapter 5), and peer ranking schemes (see Chapter 6).

Since the main goal of the client used in this thesis was to explore the methods of P2P-based multimedia streaming, the final part of this chapter deals with the specifics of streaming multimedia and quantifying the quality parameters of streaming multimedia delivery. Here, the parameters chosen to do performance measurement were selected to be in-line with the parameters used in other works in the field.

CHAPTER 4

# Dataplane Congestion Control

Early P2P communication software used UDP as a transport layer protocol for user-plane data transmission. Since UDP is a simple transport layer protocol providing only port-based multiplexing and (rudimentary) error checking, a side effect of using early P2P communication software was "total" congestion of user's communication link. Most of the early P2P communication tools were used to download files (in contrast to streaming multimedia). However, downloading files without congestion control reduced the QoS of other network applications running on a computer. For example, browsing the Internet (which is done using TCP protocol) was nearly impossible simultaneously with a P2P files transfer.

One way to reduce congestion of a user's internet connection would be to use TCP as a transport layer protocol for P2P communication as well. However, TCP provides a number of functions that are not required in a P2P communication system, such as ordered messages delivery, error correction in transport layer, and stream based communication. Furthermore, with TCP being a connection-oriented protocol, it is not a suitable transport-layer protocol choice for a highly dynamic systems, such as P2P communication.

All the above issues lead to the creation of Lower-than-Best-Effort (LBE) transport protocols. These protocols are designed to result in lower bandwidth usage in a bandwidth-limited communication link, when the link is shared with TCP data flows. Among the several proposed and IETF standardized LBE protocols is LEDBAT. LEDBAT is designed to track one-way delay of communication link and adjust the sending rate accordingly with the aim of yielding to the TCP data flows sharing the same communication link. A standard compliant PPSPP protocol implementation must use LEDBAT as a congestion control mechanism.

However, as will be shown later in the chapter, using the recommended LEDBAT configuration values can result in a sub-optimal multimedia streaming QoS, when used to stream multimedia via PPSPP. In order to improve the quality of multimedia streaming when using PPSPP and LEDBAT, this work proposes a number of changes to both protocols.

This chapter is structured as follows: Section 4.1 describes LBE protocols in general, and Section 4.2 is used to describe LEDBAT protocol in particular. Section 4.3 is used to describe the experiments performed to discover how different LEDBAT protocol parameters impact data transmission speeds in links shared with a varying number of other data flows. Section 4.4 provides details how PPSPP protocol client was integrated

with LEDBAT and how LEDBAT parameters are dynamically adjusted in response to the streaming QoS. Section 4.5 provides experimental evaluation of the proposed method of dynamic LEDBAT parameters adjustment and the discussion.

## 4.1 Related work

Lower-than-best-effort (LBE) protocols aim to achieve a lower data sending rate than a standard TCP when sharing a bottleneck link. It should be noted that LBE must not be a specific protocol. TCP protocol with a congestion control mechanism different from that of standard TCP can also be used as an LBE protocol.

According to [100], based on the methods used to achieve LBE behavior, LBE protocols can be grouped into the following groups:

1. Delay-based protocols. These protocols achieve LBE behavior by measuring the end-to-end delay and adjusting sending the rate based on the measured delay.

2. Non delay-based protocols. These protocols achieve LBE behavior by other means than measuring the end-to-end delay. Standard TCP congestion control is a non delay-based algorithm, which reduces the sending rate by reacting to loss of data.

3. Upper-layer protocols. These protocols can run "on-top" of other protocols and adjust the sending rate in the application layer, based on some observations of the system functioning.

4. Network-assisted approaches. These approaches require changes in the network infrastructure along the data path to work. For example, active queue management (ACQ) and IntServ/DiffServ based prioritization can be used to achieve LBE behavior.

TCP Vegas[111] is one of the first delay-based protocols that exhibits LBE behavior when compared to a standard TCP. It uses Round-Trip Time (RTT) measurements to calculate the expected throughput and then compares it to the actual throughput. The congestion window size is then adjusted linearly, based on the ratio of expected and observed speeds. When TCP Vegas is sharing a congested link with the standard TCP, in some cases TCP Vegas flows can be fully spaced-out by the standard TCP.

Other protocols that modify the standard TCP congestion avoidance algorithm to exhibit LBE behavior are TCP Nice[112] and TCP Low Priority[113]. To avoid being spaced out from a link by the standard TCP, TCP Nice halves its Congestion Window (CWND) at most once per RTT when Acknowledgment (ACK) loss occurs. For as long as no losses occur, TCP Nice follows TCP Vegas for congestion avoidance, by performing linear CWND adjustment. In contrast to both TCP Vegas and TCP Nice, TCP Low Priority uses One-Way Delay (OWD) instead of RTT measurements to detect congestion.

Other TCP based protocols using delay measurements to achieve LBE behavior are Sync-TCP[114], New Vegas[115], FAST TCP[116], and CODE TCP[117].

One of the non-delay based protocols is 4CP[118]. Instead of using delay measurement to gauge the delay, it counts the number of loss events and adjusts its CWND based on the number of loss events. During a severe congestion, the congestion window is reduced to some predefined value. Once the sending rate is reduced, instead of increasing the actual congestion window, 4CP is increasing an alternative congestion window (while keeping the actual congestion window at low value). Only when this alternative congestion window reaches a predefined threshold, does the actual congestion windows starts increasing.

An upper-layer approach to achieve LBE behavior is based on the monitoring of all flows leaving specific application. In the case of Background Intelligent Transfer Service (BITS)[119] that a specific application is the whole operating system. By measuring the total amount of data traffic originating from the computer, BITS tries to infer when the load is historically low, and schedule background file transmissions to occur at these periods.

## 4.2 LEDBAT Protocol

A specification of PPSPP protocol stipulates that a protocol client must use LEDBAT as a congestion control mechanism. The following chapter provides a brief description of the protocol. For reference, a brief description of TCP's congestion control mechanism is given in Appendix B.

LEDBAT uses a "scavanger" congestion control mechanism. It tries to use as much of bandwidth as is available. At the same time, it is designed to yield quickly to other data flows sharing the congested communication link. This makes LEDBAT well-suited for background data transfer applications, used to transfer large files (Apple is using a variant of LEDBAT for OS updates delivery[120]).

LEDBAT uses a congestion window to control the amount of data sent into the network in one RTT. The size of CWND can be tracked in bytes or packets. In addition, each data segment transmitted over the network using LEDBAT is prefixed with a time-stamp, which is used to measure the OWD. Upon receiving a packet, a receiver calculates the delay as a difference between the local time and a time-stamp in the data packet. The calculated delay value is then sent back to the sender in the ACK packet.

On the sender side, LEDBAT uses the algorithm shown in Figure 4.1 to calculate the CWND. Here, the target is a maximum queuing delay that LEDBAT can introduce in the network and GAIN determines the rate of CWND adjustment in response to queuing delay changes.

The above algorithm is divided into three parts. The first part is concerned with tracking of queuing delays and ensures that stale delay data is removed when no data transfers are taking place. The second part is a linear controller, adjusting the CWND value with regards to a difference between the observed and target delays. Finally, the

**function** ON_ACK(ack)
$update\_base\_delay(ack.delay)$
$update\_current\_delay(ack.delay)$

$queuing\_delay \leftarrow FILTER(current\_delays) - min(base\_delays)$
$off\_target \leftarrow (TARGET - queuing\_delay)/TARGET$
$cwnd \leftarrow cwnd + GAIN * off\_target * bytes\_acked * MSS/cwnd$
$MAX\_cwnd \leftarrow flightsize + ALLOWED\_INCREATE * MSS$

$cwnd \leftarrow min(cwnd, MAX\_cwnd)$
$cwnd \leftarrow max(cwnd, MIN\_CWND * MSS)$
$flightsize \leftarrow flightsize - bytes\_acked$
**end function**

Figure 4.1: CWND calculation algorithm in LEDBAT sender

third part constrains the CWND value to be in the range from MIN_CWND * MSS and MAX_CWND.

The method of one-way queuing delay calculation is explained using Figure 4.2. Here, a sender and a receiver are connected using a communication link having two queues (buffers). The sum data transmission delay will consist of three integral parts: propagation delay, processing delay and queuing delay. The first two parts (propagation delay and processing delay) are considered here to be constant and unique for each communication link, and the only variable part is the queuing delay.

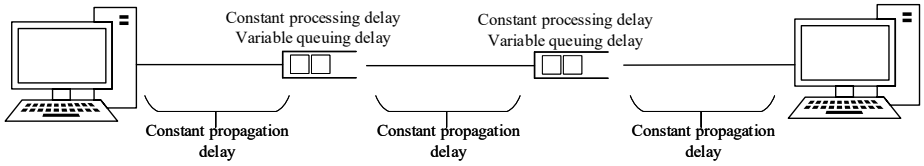



Figure 4.2: Two computers connected with a communication link containing two buffers. Propagation and processing delays are constant in this communication link.

In order to calculate the queuing delay, when only a sum delay is known, the following procedure is used. First, a base delay is calculated as a minimum value over several measurements. This base delay will be considered to be equal to the sum of propagation and processing delays. Once the base delay is known, then queuing delay can be calculated as a difference between the observed delay value and the base delay. It should be noted here that it is not required for clocks on the sender and receiver sides to be synchronized. Since the difference, and not the absolute value, is taken, any difference in clock times

cancels out when calculating queuing delay.

When a data loss (buffer overflow) occurs, LEDBAT uses the following formula to calculate new CWND value:

$cwnd \leftarrow min(cwnd, max(cwnd * 0.5, MIN\_CWND * MSS))$

It implements the multiplicative decrease part of congestion control (divides the CWND value by half), while at the same time ensuring that it does not fall bellow the minimal value.

If no ACKs are received within the Congestion Timeout Window (CTO), it is taken as an indication that either extreme congestion is occurring, or there is a significant RTT change. In both cases, the CWND is reduced to a single MSS value and the CTO value is doubled:

$cwnd \leftarrow 1 * MSS$

$CTO \leftarrow 2 * CTO$

From the protocol description above, it can be seen that the size of the CWND is influenced by four factors. First and foremost, CWND will be influenced by the ratio of the target and actual delays. Second, the size of CWND will be clamped by the MAX_cwnd value, which is directly related to the current flight-size (number of bytes or packets sent and not yet acknowledged). Third, when a loss occurs, the size of CWND will be reduced by a factor of 2. Finally, during a significant RTT change, that size of CWND will be reduced to MSS.

In order to adjust the "aggressiveness" of the LEDBAT protocol with regards to other data flows sharing a bottleneck link, two parameters can be changed—the target delay and the CWND reduction factor used when data-loss occurs. The MAX_cwnd value depends more on the flight-size, which is influenced by the connection speed. Finally, the extreme congestion occurs rarely, and hence the CWND should not be set to MSS very often. This conclusion was also observed during the experiments.

## 4.3 Experimental LEDBAT Parameters Investigation

As mentioned earlier in the chapter, all specification-compliant PPSPP clients must use LEDBAT for congestion control. However, the specification of LEDBAT proposes protocol configuration values that make LEDBAT data flows yield to other data flows. This has a negative impact on the quality of multimedia streaming, as is shown later in the experimental evaluation section. Furthermore, experiments in this chapter are designed with the notion that the user streaming multimedia (in contrast to downloading it) prefers to see an uninterrupted multimedia stream. For this reason, a number of experiments were performed to investigate how LEDBAT protocol parameters can be tuned to make it more suitable (i.e. yield less to other data flows) for use with multimedia streaming.

In order to investigate how target queuing delay and the CWND reduction factor influences the data transmission speed, a number of experiments were performed, as described next. The experiments used the following setup. Two computers were connected via a router using Ethernet connections, thus forming a dumbbell network topology. The router was used to perform bandwidth limiting using token-bucket bandwidth shaping. The token-bucket bandwidth shaper was configured to achieve an average 10 Mbps data transfer rate.

Each of the two computers ran iPerf3[121] software to generate traffic sent over the TCP connections. At the same time, both computers ran custom software, having identical functionality to iPerf3, but using LEDBAT to transfer data. The source code of the test software and the LEDBAT protocol, both implemented in Python3, are available under free and open-source license at `https://github.com/justas-/pyledbat`. Table 4.1 lists the remaining test parameters.

Table 4.1: LEDBAT evaluation experiment parameters

| Name | Values |
|---|---|
| Computers OS Kernel | Linux 4.4.0-116-generic |
| Number of TCP data-flows | 0; 1; 3; 5 |
| Number of LEDBAT data-flows | 1; 5; 10 |
| TCP flows speed | 2 Mbps; Unlimited |
| Target Delay | 10 ms; 20 ms; 40 ms; 80 ms; |
| On loss multiplier | 0.5 ; 0.75 ; 0.98 |
| Average link delay | 0 ms; 20 ms; 40 ms; 60 ms; 100 ms |
| One test duration | 60 seconds |

Tests with all combinations of these parameters were repeated 5 times. This resulted in a testing campaign consisting of 7200 tests.

### 4.3.1 Evaluation Results

The results of the experimental evaluation using two computers and a hardware router are split into two parts. The first part shows the utilization of the link by LEDBAT data flows based on the fraction of CWND that is reduced when data loss is detected. Figure 4.3 shows the results when the LEDBAT target delay is 40 ms, and Figure 4.4 shows the results for the target delay of 80 ms. Each figure differs by the number of concurrent TCP and LEDBAT data flows between the source and destination. In all figures, the link utilization is shown based on the 10 Mbps link.

The results show four main outcomes. First, the utilization of the link by LEDBAT data flows is higher, when the CWND multiplier is higher. This is an expected result,

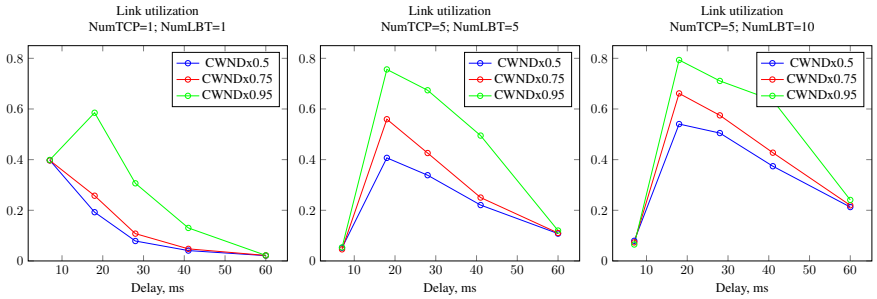



Figure 4.3: Average link utilization for different CWND multiplier values and number of TCP/LEDBAT connections. Target delay = 40 ms.

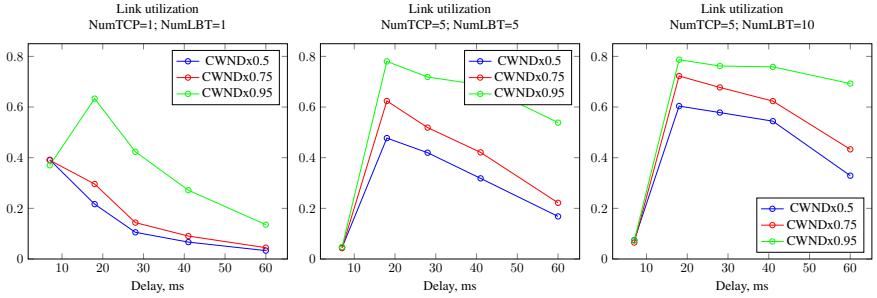



Figure 4.4: Average link utilization for different CWND multiplier values and number of TCP/LEDBAT connections. Target delay = 80 ms.

because using a higher CWND multiplier implies a slower decay of CWND value, when data loss occurs. This results in higher data sending rate.

Second, when the link is shared by two connections (one LEDBAT and one TCP), LEDBAT data flow yield to TCP data flow. Even when the loss multiplier is 0.95 (only 5% reduction of CWND on data loss), LEDBAT takes no more than 60% of total link bandwidth. However, when a number of connections is higher (5 TCP connections and 5 or 10 LEDBAT connections), TCP flows yield to LEDBAT data flows.

Third, in all experiments, the share of LEDBAT bandwidth peaks when the queuing delay is about 18 ms. As the queuing delay increases further, the share of bandwidth used by LEDBAT decreases. However, the same applies to the TCP data flows as well.

Finally, the results indicate that using a higher target queuing delay value (80 ms vs 40 ms) allows to obtain a larger share of the link's bandwidth.

The second part shows the share of link bandwidth used by LEDBAT data flows for different LEDBAT target queuing delay values. Figure 4.5 shows the results for a

CWND multiplier of 0.5; Figure 4.6 shows the results for a CWND multiplier of 0.75; and Figure 4.7 shows the results for a CWND multiplier of 0.95. As with previous results, each subfigure shows the results for a different number of TCP and LEDBAT connections.

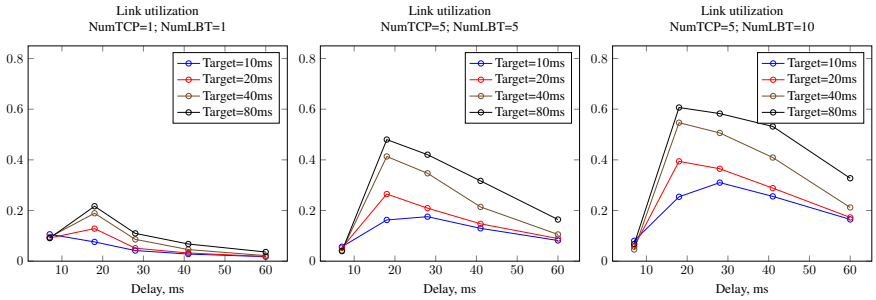



Figure 4.5: Average link utilization for different target delay values and number of TCP/LEDBAT connections. CWND multiplier=0.5

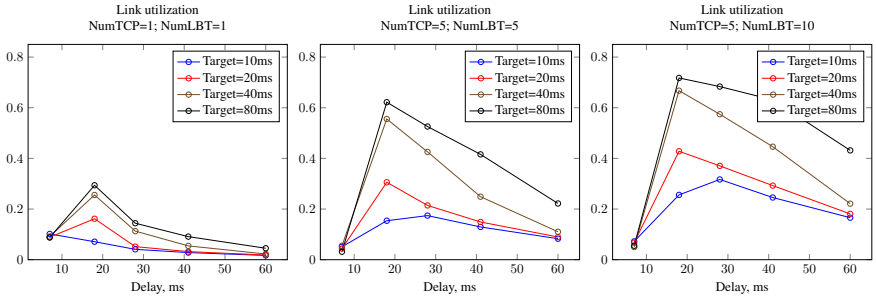



Figure 4.6: Average link utilization for different target delay values and number of TCP/LEDBAT connections. CWND multiplier=0.75

The results shown above closely follow the already-discussed results describing the impact of CWND multiplier on the bandwidth share. However, two outcomes must be noted. First, when a link is shared by only two connections, LEDBAT flow still yields to the TCP flow, event when a target queuing delay is large. Second, when data flows operate over links with high delays and high delay variability (such as mobile broadband connections, c.f. Chapter 8), a high target delay parameter does not guarantee that most of the link will be used by LEDBAT data flows. For example, consider the second subfigure in Figure 4.6. Here, when a delay variability is high (simulated queuing delay of 60ms), LEDBAT flows with target delays ranging from 10 to 80 ms use no more than 22% of link bandwidth. For comparison, when a simulated queuing delay is 18 ms, LEDBAT data flows can use up to 62% of the link's bandwidth.

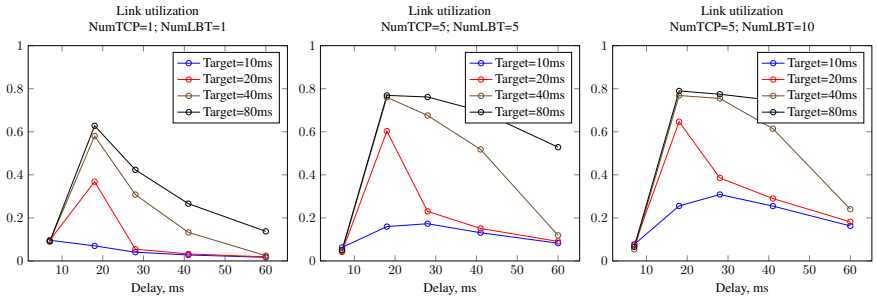



Figure 4.7: Average link utilization for different target delay values and number of TCP/LEDBAT connections. CWND multiplier=0.95

## 4.4 Using Boosted LEDBAT with PPSPP

Once the experiments using two computers were completed, a second set of experiments was designed to test the performance of multimedia streaming using different LEDBAT configuration values. For these tests, two sets of LEDBAT parameters were used, referred here as conventional LEDBAT (CLBT) and boosted LEDBAT (BLBT). The boosted LEDBAT used a target delay of 80 ms, which is under the 100 ms maximum value recommended in [95]. The boosted LEDBAT also used 0.75 as a value of CWND reduction fraction.

Initially, the experiments were envisioned in a way that all clients would use BLBT. However, using BLBT for all clients in a P2P system would be unfair. This stems from the fact that the user deriving the benefits of using BLBT and the user bearing the costs are not the same. For example, consider a situation where there are only two clients (A and B) and a multimedia streaming server, as shown in Figure 4.8. Client A starts streaming before Client B. When client B arrives, it can request data from both client A and the multimedia server. If both users were using BLBT, then all data flows from client A to the Internet (but not client B) would yield to the data-flows from from client A to client B. This way, client B would be deriving benefits (increased streaming QoS) at the cost of client A's data flows to the Internet.

One of the possible ways to solve this issue is to dynamically switch between CLBT and BLBT. This is illustrated using the example given in Figure 4.9. In the figure, PC1 starts experiencing sub-par streaming quality (Playback Continuity Index $< 0.9$). In response, it sends a request to all connected peers to switch to BLBT. One of the peers that receives this request is PC2. Before granting the requests and switching to BLBT for connection between PC2 and PC1 it checks all other outgoing connections from PC2. If at least one connection is not using BLBT, it indicates that the PC2's network link is not congested, and the request to switch to BLBT is granted.

The evaluation of multimedia streaming performance was performed in a virtual

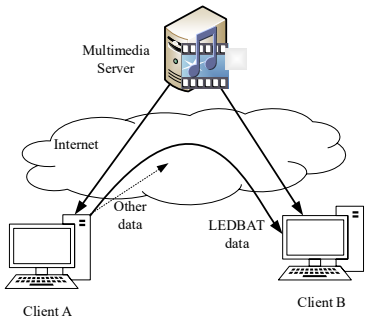



Figure 4.8: Data flow from client A to the internet would yield to the data flow from client A to client B if both clients would use BLBT.

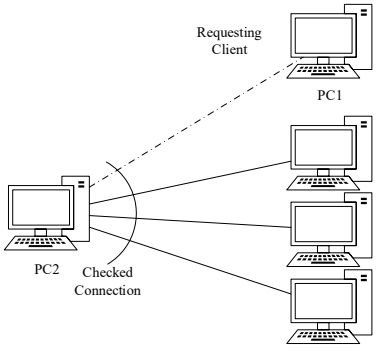



Figure 4.9: PC1 requests PC2 to switch to BLBT. PC2 does so only if at least one of the other connections is not using BLBT.

environment using the CORE networks emulator [122]. CORE divides the workstation running experiments into a number of virtual instances. Each instance represents one user performing multimedia streaming. In all experiments, the multimedia recording is a 1280x720 (HD 720p) video clip encoded with a VP8 codec having a bit-rate of 2 Mbps.

The tests used the test network topology adapted from an industry's whitepaper [123] and shown in Figure 4.10. The test network consists of three multimedia servers, a set of access and core routers and three groups of users. The tests were conducted with 5, 10, and 15 users in each user network. The links between the devices were provisions with 25 Mbps of bandwidth.

To make test conditions realistic and to vary the network utilization, the experiments were conducted using different background data traffic levels. In the base scenario ("No

Traffic"), no background data traffic was used. The normal data-traffic level ("Low Traffic") was obtained from recent research [124], indicating that 95% of home-users transfer up to 64.27 MB of data during their 15 min peak usage interval. The share between the download and upload traffic was 85% to 15% [125]. In the high background traffic scenarios ("High Traffic"), background data traffic levels were doubled.

In the tests, two user arrival patterns were used. In the "Exponential" arrival tests, user inter-arrival time followed the exponential distribution with a mean of 5 seconds and the value limited to 10 seconds. In the "Flash" arrival scenarios all users started sequentially, one-by-one.

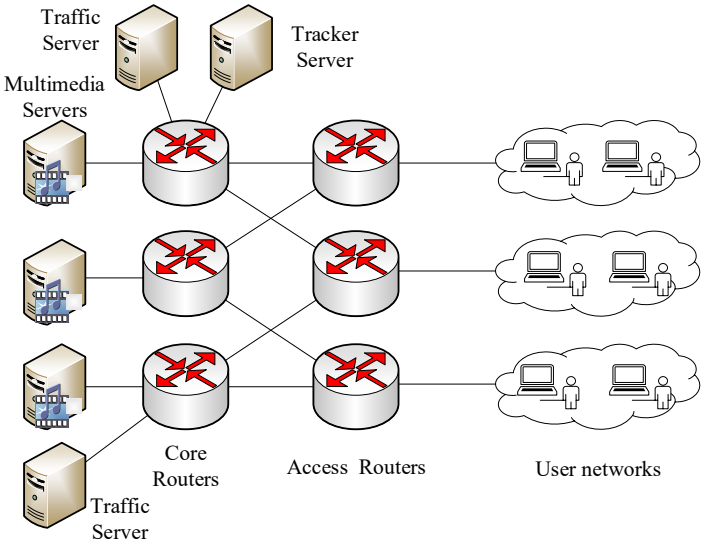



Figure 4.10: Test network topology. Users are divided into 3 groups, each connected to an access router. Each access router is connected to two core reouters. Core routers connect to multimedia servers and traffic generation servers.

## 4.5 Integration Results

The average start-up time comparison, when using conventional and boosted LEDBAT are shown in Figure 4.11–4.12. The results show that by using boosted LEDBAT, the average start-up time can be reduced by 26.3% when the average queuing delay is 26 ms, and by 24.8%, when the average queuing delay is 46 ms. In addition to significantly lowering the start-up time, it is important to note, that the start-up time was lowered in all

experiments. The reduction can be seen in all three background traffic levels, during both types of user arrival patterns and queuing delays.

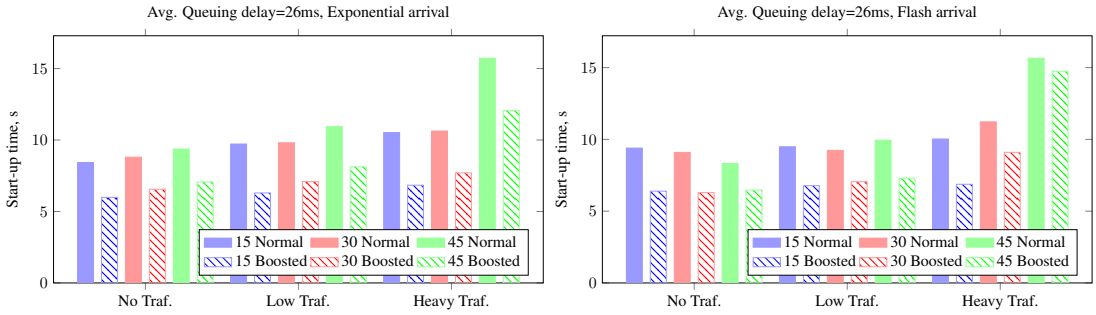



Figure 4.11: Average observed start-up time for different arrival patterns, user population sizes and background traffic levels. Numbers in the legend indicate number of users. Average Queuing delay=26 ms.

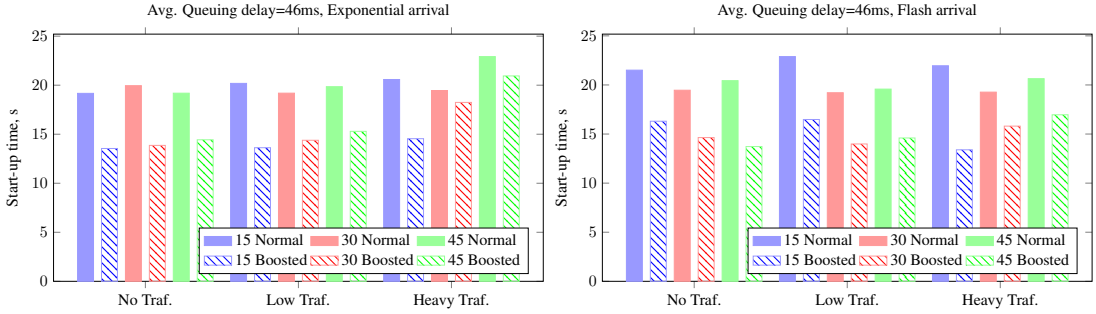



Figure 4.12: Average observed start-up time for different arrival patterns, user population sizes and background traffic levels. Numbers in the legend indicate number of users. Average Queuing delay=46 ms.

The average observed PCI when using conventional and boosted LEDBAT are shown in Figure 4.13–4.14. By using boosted LEDBAT, the average PCI can be increased by 0.3%, when the average queuing delay is 26 ms, and by 4.7%, when the average queuing delay is 46 ms. The biggest improvement of PCI by 5.2%, was observed in the experiments with an exponential user arrival pattern and 46 ms average queuing delay.

In contrast to the start-up time evaluation results discussed previously, the increase of the playback continuity can be observed only when the queuing delay is large, and the users arrival pattern has a negligible influence. This cause of this is the way TCP and LEDBAT operated in the data-links with high latency. TCP data-flow sending rate is

significantly reduced when operating in high latency links. At the same time, this latency is lower than a target queuing delay of LEDBAT (which is 80ms here), meaning that LEDBAT's congestion window size and the sending rate is not reduced.

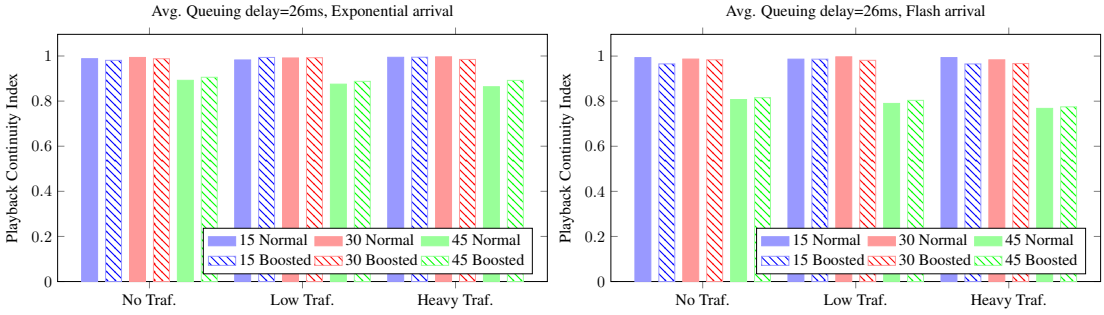



Figure 4.13: Average observed Playback Continuity Index for different arrival patterns, user population sizes and background traffic levels. Numbers in the legend indicate number of users. Average Queuing delay=26 ms.

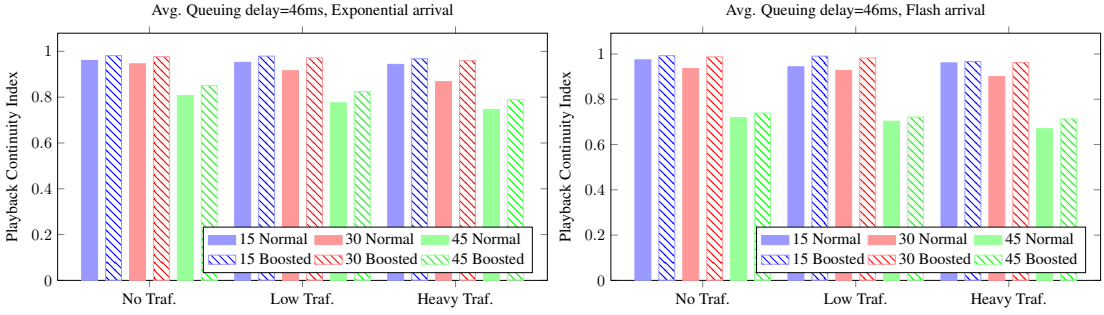



Figure 4.14: Average observed Playback Continuity Index for different arrival patterns, user population sizes and background traffic levels. Numbers in the legend indicate number of users. Average Queuing delay=46 ms.

## 4.6 Summary and Discussion

The specification of LEDBAT requires all protocol-compliant implementation to use LEDBAT as a congestion control protocol. However, the LEDBAT configuration values, suggested by its specification, are meant for non-intrusive background data transfer services. The work presented in this chapter investigated the possible changes to the

standard LEDBAT configuration values in order to make LEDBAT more suitable for multimedia streaming.

As LEDBAT is implemented in the application layer, it makes it easy to experiment with different configuration values or to change values in real time. The test results described in this chapter indicate that both the start-up time and playback continuity can be increased by dynamically changing the target queuing delay value and CWND reduction fraction. While the decrease in the start-up time is experienced in all scenarios, the increase of the playback continuity is seen only in the tests with large simulated queuing delays. While further testing is required using real (instead of emulated) clients, the results show that the the highest benefit of using boosted LEDBAT would be for users having Internet connection with high and varying latency, such as mobile broadband.

The tests described in this chapter switched between two sets of LEDBAT parameters. The switch to a faster and more "aggressive" version of LEDBAT was triggered by a fall in the playback continuity level. It remains an open question for further experimentation, if even higher playback continuity values can be achieved by adjusting LEDBAT configuration values continuously.

CHAPTER 5

# Data-Requesting Algorithms

P2P-assisted streaming is a cooperative communication system. When users download data, they make the downloaded data available for other users to download from them as well. By doing so, they reduce the communication load on the multimedia streaming server. In addition, they potentially reduce the communication load on the network itself, because connection to another user might traverse fewer network links than a connection to the streaming server.

The way a single client distributes data requests is governed by the data-requesting algorithm. In the simplest case, when there is a single user and a streaming server, all requests are sent to the server. However, the situation is more complex when there are several other users. The data-requesting algorithm must divide the requests based on the availability of data at each client.

The policy of distributing data requests depends on data being shared and the design goals of the algorithm. When P2P communication is used to share non-streaming data, such as program files, the goal might be to maximize the overall download speed. Another goal might be to ensure the highest availability of data by downloading the rares data pieces first.

In the case of streaming multimedia, the goals are different. While the overall speed is important, it is equally important to ensure uninterrupted delivery of data to prevent buffer underruns and consequentially multimedia rendering stalls. Achieving short start-up time is also important when designing a data-requesting algorithm. In the following chapter, a data-requesting algorithm for use with the PyPPSPP client is proposed. Different configurations of the proposed algorithm is tested in the simulated network to find the most optimal configuration.

The rest of this chapter is structured as follows: Section 5.1 presents some of the design goals of data-requesting algorithms used in other P2P communication clients. Section 5.2 presents the proposed algorithm. The results of the experimental evaluation and the discussion is given in Section 5.3. The chapter concludes with the summary in Section 5.4.

## 5.1 Requesting Algorithms in Literature

There are several different data-requesting algorithms proposed and analyzed for use in P2P communication systems. This section provides a brief overview of selected

algorithms based on their relevance or widespread use.

BitTorrent was originally designed for fast and efficient files transfer. The data-requesting algorithm used in BitTorrent ([126], [127]) was tuned specifically for this purpose. It combines three different policies, as described next.

The majority of the data pieces are downloaded using the rarest first policy. This policy ensures that the client selects pieces located in the least number of other peers. This policy allows to "spread" the file fastest among the users. This policy also aims to maximize the overall data transfer speed. The more peers have the piece, the higher the sum data transfer rate (over all peers) will be. Finally, the rarest first selection policy reduces the chance that some small number of data pieces will be missing, preventing the file from being completely downloaded.

The BitTorrent protocol aims for users to upload as much data as they download (i.e. to be fair and prevent free-riding). However, when new users starts downloading a file, they do not have anything to contribute with. For this reason, at the start of the file download, random pieces will be selected. This is because rare pieces (as would be selected by the rarest first policy) tend to be slower to download.

Finally, before the end of the download process, the client transitions into the end-game mode. When only several pieces are missing (typically 5), the request for these pieces are sent to several peers simultaneously. As soon as all the pieces are downloaded, the outstanding requests are canceled. While this approach can waste some bandwidth, it is usually not much.

The BitTorrent's data-requesting algorithm works especially well for files. However, it is not well-suited for multimedia streaming. By trying to download the rarest pieces first, a client often experiences playback buffer underruns, which cause stalling of multimedia playback. One way of improving the selection algorithm was proposed in [81], called BitTorrent Streaming (BiToS).

BiToS works by dividing all the outstanding data pieces into two sets – a high priority set (pieces required soon) and the remaining pieces set. The size ratio of the sets can vary. Then during the multimedia streaming, chunks are selected either from the high priority set (with probability $p$), or from the low priority set (with probability $1 - p$). Pieces in the high priority set are downloaded sequentially, and pieces from the remaining set are downloaded using the BitTorrent's rarest first policy.

The experimental evaluation of the BiToS method discovered that the best results are obtained when $p = 0.8$. In such configuration, the playback continuity index was 0.97, compared to 0.82, when BitTorrent's conventional algorithm was used.

The ideas behind the BitTorrent and BiToS algorithms can be found also in the algorithms provided by the libswift project. Libswift is a reference implementation of the PPSPP protocol. However, the PPSPP protocol itself does not define the data selection algorithm.

Libswift provides 4 different algorithms[128], called pickers:

1. Sequential picker. This algorithm downloads data pieces sequentially with a small amount of randomization.
2. Live picker. This algorithm is designed for use with Live streaming data. It downloads data pieces sequentially. However, it starts not at the beginning of the stream, but at the point the client "tunes-in" into the live stream.
3. Rarest-first picker. This picker implements the rarest-first strategy as described previously.
4. Video-on-Demand (VoD) picker. This picker divides the multimedia stream into three ranges using different thresholds. The highest priority (closest to the playback point) set is downloaded sequentially, the other two sets are downloaded using the rarest-first strategy.

Based on the findings from the above papers, BiToS algorithm would be a good choice for use in the PyPPSPP client. However, BitTorrent operates on a chunks with the size ranging from 32 KB to 16 MB, while the PyPPSPP client operates on 1 KB size chunks. Porting the BiToS algorithm to PyPPSPP would significantly increase the algorithm run-time. The number of data chunks for the same size multimedia file in PyPPSPP is 32-16384 times larger than in BitTorrent due to the smaller chunk size. Calculating "rareness" of each chunk would be resource prohibitive. Due to this reason, the proposed algorithm described next omits the rareness calculation and operates sequentially exclusively at around the multimedia playback point.

## 5.2 Data-Requesting Algorithm

The following section describes an algorithm implemented in the PyPPSPP client that sends requests to other peers to send multimedia data to the requesting peer. Before discussing the actual algorithm, this section describes relevant information and terms used in the algorithm description.

As described previously in Chapter 3, information shared using PPSPP protocol is divided into chunks. Chunks can have different sizes between different protocol implementations, but all clients must use the same chunk size when communicating in the swarm. PPSPP implementation used in this thesis used 1 KB data chunks. All chunks are numbered sequentially, starting at 0.

Each client maintains a set (data structure) containing chunk numbers that the client has already downloaded. Every client advertises the contents of this set periodically to connected peers via the HAVE messages. Additionally, each client maintains two sets for each connected member. The first set (*have set*) contains chunks numbers that the remote peer advertises (via the aforementioned HAVE messages). The second set (*outstanding set*) contains chunk numbers that the client requested from the remote peer, but has not received yet.

When a data chunk is received by a client and it passes the integrity verification (if performed), the following actions take place. The chunk number is removed from the outstanding set of the peer that sent the chunk. Then the same chunk number is added to a clients set containing chunks present locally. At some time in the future, the client will update the HAVE messages it sends with information about the just-received chunk.

The data-requesting algorithm can maintain (if configured) two chunk selection windows, which limit what chunks can be requested. If the client is configured to discard chunks after playback (has the discard window), then the data-requesting algorithm will not request chunks that have a smaller number than the number of the chunk being rendered now less the discard window. In the same way, if the client is configured with the forward downloading window, then no chunks will be requested that have a number higher than the chunk number being rendered now plus the download forward window value.

Finally, some chunk sets can be calculated on-demand by the client each time they are needed. In order to get a set of all requested but not received chunks, a logical union operation is performed on all outstanding sets of connected peers.

### 5.2.1 Algorithm Implementation

The pseudo-code of the data-requesting algorithm, as implemented in the PyPPSPP client is shown in Figure 5.1. The algorithm can be divided into several blocks, each described next.

At the start of each algorithm run, a set of all outstanding chunks (`set_all_outstanding`) is computed. This is done by looping over the peers list and applying union operation on the each peer's outstanding chunks set. This outstanding chunks set is calculated only once at the start of the run, and then later is only updated by adding chunks requested during an algorithm run.

Once the set of outstanding chunks is calculated, the algorithm shuffles the local peers list. Since the algorithm will be looping over the peers list, by shuffling the list beforehand, an imperfect (but satisfactory) load sharing is performed among the peers. The algorithm continues by evaluating each peer in the now shuffled peers list.

The evaluation starts by comparing the cardinality (number of items) of the peer's outstanding set to a threshold value (`THRESH_OUTSTANDING`). If the number of outstanding chunks is higher than a threshold, the peer is ignored in this algorithm run, and the algorithm continues with the next peer.

If the number of outstanding chunks is less than a threshold, the algorithm computes a set of candidate chunks (`peer_candidates`), that the peer has and the client might be interested in having. This is done by subtracting the set of chunks already available in the client from the set of chunks that the peer has. The resulting set is further reduced, by removing all chunks that are already outstanding. The resulting set is filtered twice, by applying the discard window and forward download values. The resulting set (`required`) contains the chunk numbers that the client might try to request.

```
function REQUEST_DATA
    for peer in peers_list do
        set_all_outstanding ← set_all_outstanding + peer.outstanding_chunks
    end for

    shuffle(peers_list)
    for peer in peers_list do
        if peer.set_outstanding.lenght > THRESH_OUTSTANDIG then
            continue
        end if

        peer_candidates ← member.set_have - client.set_have - set_all_outstanding
        required ← filter(peer_candidates, discard_wnd)
        required ← filter(required, forward_wnd)

        if len(required.length > THRESH_REQUEST) then
            required.sort()
            set_req ← required[0:THRESH_REQUEST]
            peer.request(set_req)
            set_all_outstanding ← set_all_outstanding + set_req
        else
            peer.request(required)
            set_all_outstanding ← set_all_outstanding + required
        end if
    end for
end function
```

Figure 5.1: Data chunks selection algorithm integrating ALTO cost metrics

The final part of the algorithm ensures that the number of chunks requested from the peer is less than the request threshold (`THRESH_REQUEST`). If the cardinality of the required set is less than a request threshold – this whole set is requested from the peer. If the cardinality is more than the threshold value, then up to the threshold value of lowest numbered chunks is requested. Finally, the set of all outstanding chunks is extended to include the just-requested chunks.

## 5.3 Experimental Algorithm Evaluation

As can be seen from the description of the algorithm, it tries to spread the load among the clients, by not requesting more than `THRESH_REQUEST` number of chunks from each client. Furthermore, it does not increase the amount of chunks outstanding by the client,

if the number of outstanding chunks is already above `THRESH_OUTSTANDING`.

The goal of the experimental evaluation is to measure how the start-up time and the PCI changes based on the values of the above-mentioned thresholds. In order to do so, the experiments reused the setup described in Section 4.4. The only change from the initial setup was the omission of test with no background traffic. Furthermore, all clients used the boosted LEDBAT scheme, as described in Chapter 4.

The summary of the test parameters are give in Table 5.1

Table 5.1: Data-requesting algorithm test parameters

| Name | Values |
|---|---|
| Number of clients | 15; 30; 45 |
| Background traffic levels | Low traffic; Heavy traffic |
| Clients arrival pattern | Flash; Exponential |
| THRESH_REQUEST values | 100; 200; 300; 400 |
| THRESH_OUTSTANDING values | 50; 150; 250; 350 |
| Number of repetitions | 5 |

Here it should be noted that in the experiments, the THRESH_OUTSTANDING value was never higher than the THRESH_REQUEST value. For example, in the experiments using 200 as a THRESH_REQUEST value, THRESH_OUTSTANDING was set to 50 and 150.

### 5.3.1 Evaluation Results

The average start-up times observed during the experiments are shown in Tables 5.2–5.4. The tables indicate an average observed start-up time, based on the background traffic level. "Req" (rows) and "Out" (columns) indicate request threshold and outstanding threshold values used in the data-requesting algorithm. In the start-up time observations, the pattern of users arrival had negligible influence ($< 1\%$ difference from the average), hence only the average values for both arrival patterns are shown.

The results indicate that the start-up time is influenced by both the requesting and outstanding threshold values. In all experiments, higher threshold values directly correspond to longer-start-up times (the possible reasons are discussed later). The only two exceptions to this observation are experiments with low traffic levels and threshold values of 100/50. In these two cases, the start-up time is longer than the start-up time of a client using larger algorithm thresholds. This might indicate, that in certain cases, the smaller threshold value is actually not beneficial.

The results also clearly show that the increase of the background traffic levels has a negative effect on the start-up time. The start-up times in the tests with high background traffic levels are on average longer by 10% (for clients using small threshold values).

Table 5.2: Start-up time, seconds (15 users)

| Traffic | Low | | | | Heavy | | | |
|---|---|---|---|---|---|---|---|---|
| Out Req | **50** | **150** | **250** | **350** | **50** | **150** | **250** | **350** |
| **100** | 9.50 | | | | 10.40 | | | |
| **200** | 10.46 | 10.11 | | | 11.01 | 10.43 | | |
| **300** | 10.71 | 12.83 | 12.74 | | 11.15 | 11.50 | 12.76 | |
| **400** | 11.20 | 11.30 | 12.93 | 13.88 | 11.42 | 11.56 | 13.61 | 15.28 |

Table 5.3: Start-up time, seconds (30 users)

| Traffic | Low | | | | Heavy | | | |
|---|---|---|---|---|---|---|---|---|
| Out Req | **50** | **150** | **250** | **350** | **50** | **150** | **250** | **350** |
| **100** | 8.50 | | | | 11.66 | | | |
| **200** | 9.05 | 10.67 | | | 11.02 | 12.20 | | |
| **300** | 9.49 | 11.49 | 11.99 | | 10.12 | 11.85 | 13.31 | |
| **400** | 9.32 | 10.56 | 12.74 | 14.50 | 11.14 | 12.28 | 14.02 | 15.80 |

Table 5.4: Start-up time, seconds (45 users)

| Traffic | Low | | | | Heavy | | | |
|---|---|---|---|---|---|---|---|---|
| Out Req | **50** | **150** | **250** | **350** | **50** | **150** | **250** | **350** |
| **100** | 10.14 | | | | 11.05 | | | |
| **200** | 8.81 | 11.16 | | | 11.66 | 14.74 | | |
| **300** | 9.34 | 11.34 | 13.04 | | 11.93 | 14.95 | 17.38 | |
| **400** | 9.23 | 11.42 | 12.85 | 14.62 | 13.19 | 15.75 | 16.57 | 19.52 |

However, the increase of the number of users has a much smaller effect on the start-up times. The average start-up times of all threshold combinations are 11.56, 10.86 and 11.16 seconds, for 15, 30 and 45 users respectively. All the values are within $\pm$ 3% of the average. This indicates that the number of users has a negligible influence on the start-up time.

Table 5.5: Playback Continuity Index (15 users)

| Traffic | Low | | | | Heavy | | | |
|---|---|---|---|---|---|---|---|---|
| Out Req | **50** | **150** | **250** | **350** | **50** | **150** | **250** | **350** |
| **100** | 1.00 | | | | 0.99 | | | |
| **200** | 0.99 | 0.99 | | | 0.98 | 0.99 | | |
| **300** | 0.99 | 0.98 | 0.98 | | 0.98 | 0.99 | 0.97 | |
| **400** | 0.97 | 0.98 | 0.97 | 0.98 | 0.98 | 0.96 | 0.97 | 0.97 |

Table 5.6: Playback Continuity Index (30 users)

| Traffic | Low | | | | Heavy | | | |
|---|---|---|---|---|---|---|---|---|
| Out Req | **50** | **150** | **250** | **350** | **50** | **150** | **250** | **350** |
| **100** | 0.99 | | | | 0.99 | | | |
| **200** | 0.99 | 0.99 | | | 0.99 | 0.99 | | |
| **300** | 0.99 | 0.98 | 0.99 | | 0.99 | 0.99 | 0.98 | |
| **400** | 0.98 | 0.98 | 0.98 | 0.98 | 0.99 | 0.97 | 0.97 | 0.97 |

Table 5.7: Playback Continuity Index (45 users, Exponential Arrival)

| Traffic | Low | | | | Heavy | | | |
|---|---|---|---|---|---|---|---|---|
| Out Req | **50** | **150** | **250** | **350** | **50** | **150** | **250** | **350** |
| **100** | 0.85 | | | | 0.84 | | | |
| **200** | 0.84 | 0.88 | | | 0.84 | 0.86 | | |
| **300** | 0.80 | 0.85 | 0.82 | | 0.79 | 0.80 | 0.81 | |
| **400** | 0.82 | 0.80 | 0.81 | 0.84 | 0.77 | 0.74 | 0.76 | 0.77 |

The average observed PCI values are shown in Tables 5.5–5.7. All the tables indicate the average observed PCI, based on the threshold values and the background traffic levels. The users arrival pattern had a smaller than 1% impact on the average PCI value in the experiments with 15 and 30 users, and hence the results are averaged for both arrival patterns. The results for experiments with 45 users are shown separately for both user

Table 5.8: Playback Continuity Index (45 users, Flash Arrival)

| Traffic | Low | | | | Heavy | | | |
|---|---|---|---|---|---|---|---|---|
| Out Req | **50** | **150** | **250** | **350** | **50** | **150** | **250** | **350** |
| **100** | 0.77 | | | | 0.75 | | | |
| **200** | 0.79 | 0.78 | | | 0.74 | 0.73 | | |
| **300** | 0.75 | 0.74 | 0.76 | | 0.71 | 0.73 | 0.74 | |
| **400** | 0.74 | 0.73 | 0.73 | 0.74 | 0.70 | 0.70 | 0.71 | 0.73 |

arrival patterns in Tables 5.7 and 5.8.

The results show that for both 15 and 30 user experiments, the number of users has no significant impact on the PCI values. The impact of the background traffic levels on the PCI value is also not significant. Throughout all 15 and 30 user experiments, the average PCI value decreased by 0.005 in experiments with the higher background traffic level.

The results of the experiments with 45 users indicate larger differences in the PCI values based on the traffic levels, users arrival pattern and threshold levels. The average PCI value falls by 0.07 when users arrive in the flash pattern, compared to the exponential arrival. The increase of the background traffic levels lowers the PCI value by 0.04 and 0.03 in exponential and flash arrival scenarios respectively. Finally, the results of the 45 users experiments clearly show that the highest PCI values are obtained when the outstanding chunks threshold values is 200 or less.

### 5.3.2 Discussion

The results described above indicate that the best results in terms of both start-up time and playback continuity are achieved when the size of of requests is small. This result might seem counter-intuitive. In the conventional communication systems, a smaller number of larger requests is usually preferred to larger number of smaller requests. This applies to a wide array of examples, from Ethernet adapters and routers to Web servers.

However, smaller requests are beneficial in P2P-assisted streaming systems in general, and for the tested system in particular. In general, by using smaller request sizes, the communication load is shared among more users. For every client, the number of data chunks that it requests up-front from the current playback position is limited by the forward download window. For a fixed size window, a smaller request threshold size means that the same amount of required chunks (forward download window size) is divided among a larger number of peers. This in turn increases the chances that the data will be requested from the peers that are "closer". This is further explored in Chapter 6, where peers are ranked based on the network connection quality before requesting data.

The increase in streaming quality when the data request size is small is also influenced by the implementation of the client. The PyPPSPP client is a single-threaded program. Furthermore, it fulfills data requests in first-in-first-out method. When the data requests are large, the client program spends more time fulfilling the request. As the program is single-threaded, other tasks, such as requesting data for client's own consumption, and processing of other data requests must wait until the completion of data sending. Overall, working with larger request sizes results in lower overall performance.

## 5.4 Summary

This chapter presents a data-requesting algorithm used in the PyPPSPP client. Data-requesting algorithms are used to divide the data requests among the communication peers - that is among the streaming multimedia server and other users. This chapter presents the different data-requesting policies, as implemented in the data-requesting algorithms used in BitTorrent, BiToS, and libswift software.

The algorithm proposed for use in PyPPSPP is extensively tested in different network conditions, including different user population sizes, background traffic levels and user arrival patterns. The results showed that the proposed algorithm achieves the highest playback continuity and the lowers start-up times when the request size is small. The possible reasons for this are identified and discussed.

While the proposed algorithm ensures high playback continuity, start-up times could still be improved. Presently, an average start-up time of about 10 seconds is too large for widespread deployment. Methods to reduce this time are left for future research. However, starting the streaming immediately from the streaming server and then connecting to other users might be one viable option. Another might be to use different data request threshold sizes, based on the fact whether the playback of multimedia started or not.

CHAPTER 6

# External Peers Ranking

Ranking of potential communication peers is important in numerous networking application areas. Clients of Content Delivery Networks (CDNs) can use ranking to find the best CDN node. Players in multiplayer games can use peers ranking to either join the server closest to them, or to form a players group consisting of players having approximately the same communication latencies. Finally, P2P communication can use peers ranking to request information from the peers with favorable connection conditions.

Generally, peers ranking can be performed by the peers themselves, or by a special external peers ranking service. Typically, when performed by the peers themselves, peers ranking is performed by evaluating the communication RTT (by using a *ping*-like tool), number of intermediate hops (by using a *traceroute*-like tool), or by evaluating the length of an AS-path (detailed in the next section).

When ranking is performed by an external third-party service, a client requesting ranking will receive either the address of the best peer, or some kind of (abstract) communication cost value assigned to each potential peer. Providing the address of the best communication peer is widely used in the Internet during the Domain Name System (DNS) lookup. For example, consider trying to resolve an IP address for the domain `google.com`. This will be done by discovering the Web servers closest to the requesting user, and then returning the IP addresses of the closest peers in the DNS reply.

The following thesis chapter explores how the external peers ranking can be used by P2P communication systems. The goal of such ranking is to perform better than random communication peers selection, and thus improve the performance of multimedia streaming. The ranking of peers is performed by the Application Layer Traffic Optimization (ALTO) server. As described in more detail further in section 6.2, ALTO is an IETF standardized protocol for exchanging information about network topology and the cost of reaching end-points in the network. The evaluation of ALTO was performed in a virtual network using a prototype implementation of the ALTO server.

In the remaining part of the chapter, the methods of communication peers ranking are discussed in section 6.1. The description of the ALTO protocol is given in section 6.2. The methods used to derive the ALTO cost values are described in section 6.3. The description of the prototype implementation of the ALTO server is given in section 6.4, along with the description of the experimental setup. The results of experimental evaluation the influence of ALTO on the performance of multimedia streaming is given in section 6.5. The chapter

concludes with the final remarks in section 6.6.[1]

## 6.1 Methods of evaluating nodes in the Internet

Peers ranking is an important part of P2P communication. Requesting data from peers that are "close" in the network allows to receive required data faster, and with less interruptions. However, peers evaluation and ranking is not a trivial task.

A naïve method to rank peers is to evaluate each one individually. There are many ways this can be done–for example, by measuring a data transmission RTT, the number of intermediate hops, or using some other metric. While this approach can be used in networks with a small number of hosts, this is not scalable in large networks. If symmetric data-connection conditions are assumed (which is not always the case in the Internet), nodes must perform $N(N-1)/2$ evaluations in a network of $N$ nodes.

One way to reduce the amount of information required to evaluate the nodes is to assign coordinates (a set of numbers) to each node. Then, connection conditions between two nodes can be approximated using a distance function over the value of coordinates of the nodes. In a system of $N$ peers, the distances to all peers can be computed by maintaining $N$ sets of records, each D numbers large (where D is a number of coordinates space dimensions), taking $O(N * D)$ amount of space. Compare this with $N(N-1)/2$ individual distances and $O(N^2)$ amount of space used in naïve approach, where each actual measurement is maintained.

One of an earliest protocols, designed to assign coordinates to the nodes in the network, is GNP[88]. In GNP, each node is assigned a coordinates value in an N-dimension coordinates space. Then, quality of a connection is estimated using distance between the nodes' coordinates. The bigger the distance – the worse the estimated connection quality.

GNP coordinates of the network nodes are determined by measuring the ping times between the nodes and a set of special landmark nodes. Landmark nodes are special nodes, placed throughout the network, that are used as a reference for all coordinate calculations. The coordinates of the landmark nodes themselves are determined by measuring the ping times between all the landmark nodes, as described next.

For an example, refer to Figure 6.1. Here, the Euclidean 2D space is used for nodes placement. The landmark nodes are shown using the red color and the remaining two client nodes are show using the green. At the start of GNP operation, the landmark nodes measure ping delays between themselves (indicated with black dashed arrows). These measurements produce a matrix, where each entry represents a round trip time between the landmark nodes represented by column and line indexes. Once the RTTs matrix is obtained, some function is used to assign the coordinate values to each node, so that the error between the nodes distance and the actual measured ping-time-delay is minimized. One of the possible functions to obtain the coordinates values might be Simplex Downhill, as proposed in [129].

[1] Contents of this chapter is based on the paper 5 from the thesis publication list.

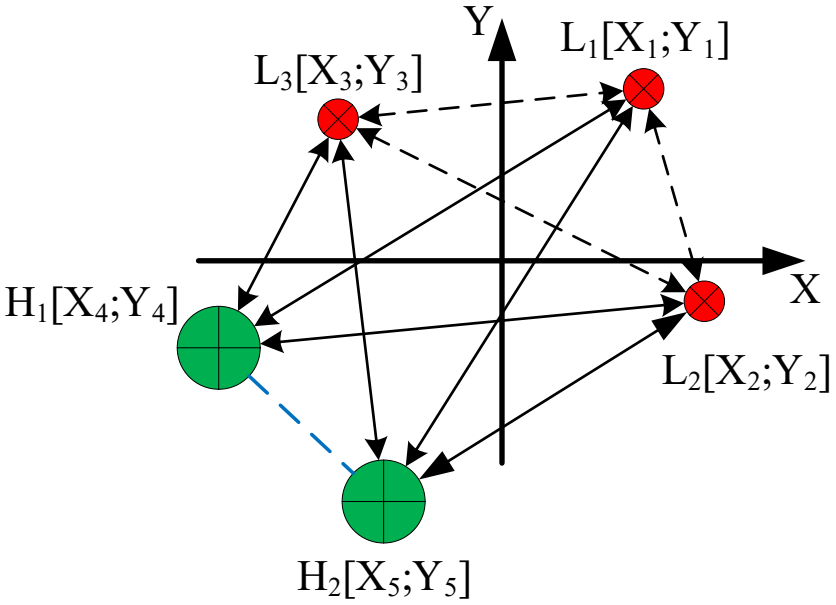



Figure 6.1: Network node coordinates establishment using GNP protocol with 3 landmark nodes.

The network nodes follow the same procedure to obtain their coordinates. First, they measure the ping times to a set of landmark nodes, as shown using solid black arrows between the nodes and landmarks in the above figure. Then, they use the same function as used by the landmark nodes to assign a coordinates value that minimizes the error between the distance to the landmark node and the related ping delay. Finally, once all nodes have coordinates values assigned, the parameters of a connection between two nodes can be estimated using a distance between the coordinates of the nodes. This is shown as a blue dashed line in the above figure.

An extended and decentralized version of the network coordinates system is proposed in a protocol called Vivaldi[89]. In Vivaldi, same as in the GNP, all nodes are assigned coordinates, and the connection quality is modeled as the distance between the coordinates. Furthermore, as in GNP, Vivaldi also uses ping RTT measurements to calculate the coordinate values.

However, instead of being static, coordinate values in the Vivaldi protocol are dynamic. The Vivaldi coordinate system is modeled as a system of physical mass-springs, where each pair of nodes is linked using a mechanical spring. Then, when assigning coordinate values to the nodes, the goal is to assign such a value that the force extended to the nodes

by the springs is in equilibrium and the system is still.

When the initial coordinates are calculated and the system is in equilibrium, any further RTT measurement will cause a change in the coordinates relative to both: the new measured value and the sum force of all springs "connected" to the node.

While the previous two proposals focused on distributed assigning coordinates to the nodes and finding a distance between the nodes to calculate connection parameters, other proposals suggest a more centralized approach. For example, Meridian[130] protocol can be used to find the closest Meridian node to any chosen target node in the network.

Meridian works as follows: each Meridian node tracks RTTs to a set of other Meridian nodes. Based on the RTT value, nodes are grouped into different groups. Each group can be visualized as a concentric circle, centered on the Meridian node as shown in Figure 6.2. The radius of all concentric circles increases exponentially.

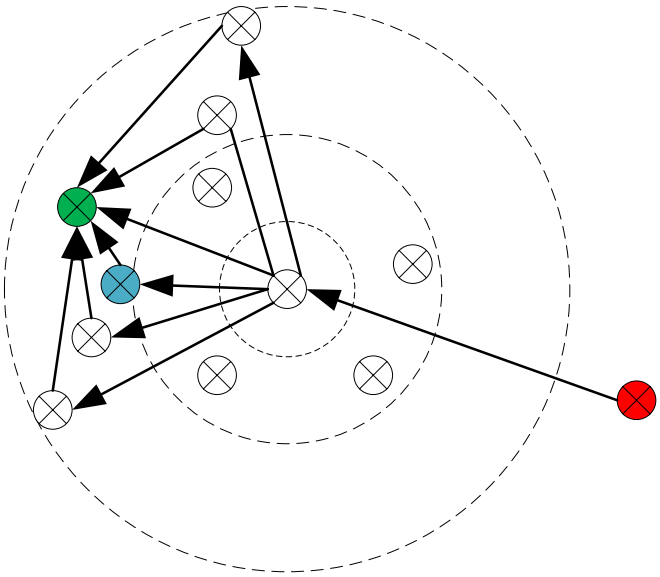


Figure 6.2: Finding the closest node using Meridian protocol

Finding the closest Meridian node to some target node works as follows. The client (shown as a red dot in the above figure) sends a request to find the closest Meridian node to the target (green node in the same figure) to a randomly chosen Meridian node (central node in the above figure). The central Meridian node measures the RTT to the target node. Then it relays the request to find the closest node to all other known Meridian nodes in the circle having the radius larger than the measured value (the same circle that encompasses the target node). Upon receiving this request, each Meridian node performs

the measurement and relays the request further on. The process stops when no Meridian node can find a relay closer to the target node. In the above example, the closes node is the blue node, which is found after a single relay.

All the above mentioned methods use RTT to estimate connection quality. However, several works in the field propose to use the length of the Autonomous System (AS)-path to rank the network nodes [131]–[134]. This method is based on the fact, that all nodes in the Internet can be mapped to the unique networks they belong to and identified by the AS number. Then, any path in the Internet between two nodes can be mapped to a list of traversed networks, identified by their corresponding AS numbers. For example, consider Figure 6.3.

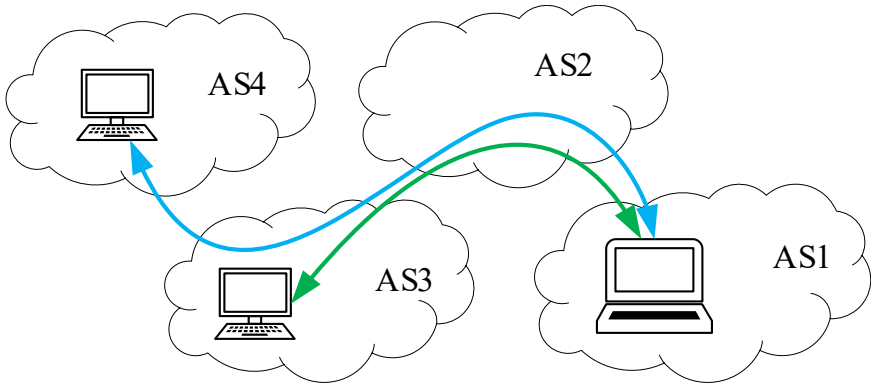



Figure 6.3: Two communication paths traversing different number of autonomous-systems.

Here, a node in network AS1 communicates with nodes in the networks with AS numbers 3 and 4. Consequently, AS-path to node in network 3 is AS1-AS2-AS3, and AS1-AS2-AS3-AS4 to the node in network 4. While this method is easy to implement, it is not as accurate as other discussed methods. By using the AS-path alone for connection estimation, the conditions of the traversed networks (such as available bandwidth and congestions) are ignored.

A network node performing evaluation using the above methods must either perform several measurements (to calculate coordinates), or to maintain a network topology database and then perform route tracing to determine intermediate network hops. These methods might not be suitable for highly dynamic situations, like P2P communication, where network nodes churn might be large. Furthermore, using the above listed methods removes the opportunity for the network operator to influence the decisions. This is a big drawback, because the network operator has much better knowledge of its internal network. The remaining part of the chapter will explore the idea of delegating the ranking of communication peers to a network operator by utilizing the ALTO protocol.

## 6.2 Introduction to the ALTO Protocol

ALTO is a data exchange protocol standardized by IETF. The goal of the ALTO protocol is to provide a standardized messages syntax and vocabulary that can be used to exchange abstract information about computer networks. ALTO protocol messages are exchanged between an ALTO protocol client and the ALTO server.

ALTO protocol specification provides a list of services that an ALTO server must provide. The only mandatory service is the *Map Service* encompassing *Network Map* and *Cost Map* sub-services. Optional services include *Map-Filtering*, *Endpoint Property*, and *Endpoint Cost* services. As the ALTO protocol is used to provide information about the network, all the above listed services use the concept of Provider-defined Identifier (PID) to identify network locations. A single PID is used to identify a collection of network endpoints. When used in large networks, PIDs can form hierarchical topologies. For this, consider the example shown in 6.4. Here, a PID representing a European-wide IP network is divided into three country-wide PIDs. Each country-wide PID is in turn divided into PIDs representing each Internet exchange point. Other PIDs, not shown in the figure, can also be present in the network.

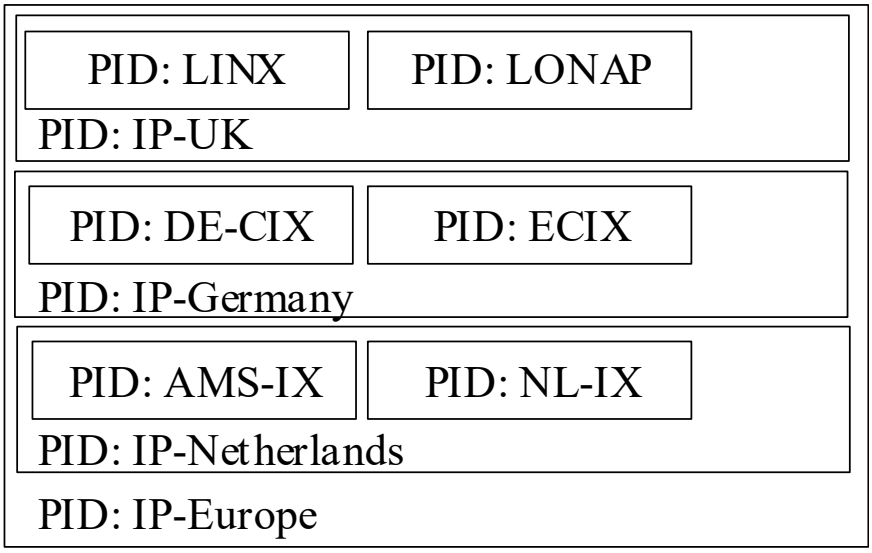



Figure 6.4: An example of a European-wide network's PIDs hierarchy.

Each PID references a range of endpoint addresses. In a general case, any addressing method can be used (for example ATM VPI/VCI or IP), however for the use-case described in this chapter, IPv4 addresses are used.

Once an ALTO server administrator creates a hierarchy of PIDs representing a network topology, then this information can be accessed using the *Network Map* service. In order to request a network map, a client makes a Representational State Transfer (REST) request to the ALTO server, to which the server replies with a JavaScript Object Notation (JSON) encoded network map. An example, showing a partial network map used for experiments described later, is shown in listing 6.5.

```
{
    "network-map": {
        "adslam-4": {
            "ipv4": [
                "192.168.4.0/24"
            ]
        },
        "core-dc": {
            "ipv4": [
                "192.168.240.0/24",
                "192.168.245.0/24"
            ]
        },
        "adslam-2": {
            "ipv4": [
                "192.168.2.0/24"
            ]
        }
    },
    "meta": {
        "tag": "5feceb66ffc86f38d952786c6d696c79c2dbc239↩
           dd4e91b46729d73a27fb57e9",
        "resource-id": "network-map"
    }
}
```

Figure 6.5: A fragment of a network map used in experiments.

The network map shown above contains three PIDs (`adslam-4`, `adslam-2`, `core-dc`), with two PIDs encompassing a single IP network each, and one PID (`core-dc`) encompassing two networks. Meta information in a network map is used to uniquely identify each map and its version.

Similarly, an ALTO client can make a request to an ALTO server to get a cost value of sending data either from one PID to another, or from one IP address to another. The cost value here defines some abstract cost of sending data one way. The cost can be dimensionless, or can represent actual monetary cost, delay, composite load or any other provider-defined metric. The ALTO standard does not define any method to derive the cost value, leaving this choice to the ALTO server operator. An example of data-routing cost lookup and the associated response is given in listing 6.6.

In the above example, a request is made to get a data-routing cost, using a numerical "hops-path" metric from IP address `192.168.245.2` to addresses `192.168.3.2` and `192.168.3.4`. The response contains a reference to the network-map, which was used to calculate the data-routing cost values. The response also contains the cost-map, indicating data routing costs of 25 and 32 to the two destination addresses respectively, indicating that sending data to IP address `192.168.3.2` is preferred over sending data to IP address `192.168.3.4`. A discussion on the methods to derive and interpret cost values is given in Section 6.3.

In addition to evaluating the connections between two endpoints in the network, the ALTO server can be used to discover information about endpoints in the network using the *Enpoint Property* service. To do so, an ALTO client sends a network property request to the ALTO server. The request contains a list of end-points and a list of the properties of the indicated end-points that the client is interested in. Upon processing the request, the ALTO server responds with a cross-product of all requested end-points with their properties. In addition to learning about end-point properties, the end-point discovery service can be used to discover end-points as well. An example end-point discovery request and response is given in listing 6.7.

In the above example, a client sends a request to discover all end-points known to the ALTO server (indicated by IP address `0.0.0.0`). The client is interested in two properties – the PID of the end-point and the presence of the `stream-cache` property. After receiving the response, the client knows the IP addresses of two streaming cache servers in the network in addition to the PIDs assigned to them. The information about the PIDs can be used to rank two possible cache servers based on the connectivity conditions to them by utilizing the already-described *Cost Map* service. The use-case of locating streaming multimedia cache servers is further explored in Chapter 8, where it is used to discover the address of the multimedia streaming server located in the mobile network base-station.

## 6.3 ALTO Data-routing Cost Metrics

As already described, the ALTO protocol specification does not define a method to derive the data-routing cost metric. Some of the works discussed previously proposes to use the length of an AS-path between the source and destination PIDs as a cost metric. However, using the AS path's length alone does not take into account the conditions of the networks

```
REQUEST:
{
  "cost-type" : {
    "cost-mode" : "numerical",
    "cost-metric" : "hops-path"
  },
  "endpoints" : {
    "srcs": ["ipv4:192.168.245.2"],
    "dsts": [
      "ipv4:192.168.3.2",
      "ipv4:192.168.3.4"
    ]
  }
}

RESPONE:
{
  "meta" : {
    "dependent-vtags" : [{
      "resource-id": "network-map",
      "tag": "5feceb66ffc86f38d952786c6d696c79c2dbc239dd↩
         4e91b46729d73a27fb57e9"
    }],
    "cost-type" : {
      "cost-mode"  : "numerical",
      "cost-metric": "hops-path"
    },
    "cost-map" : {
      "ipv4:192.168.245.2": {
        "ipv4:192.168.3.2": 25,
      "ipv4:192.168.3.4": 32
      }
    }
  }
}
```

Figure 6.6: ALTO routing cost request and response.

```
REQUEST:
{
  "properties" : [
    "networkmap.pid",
    "priv:stream-cache"
  ],
  "endpoints"  : [ "ipv4:0.0.0.0"]
}

RESPONE:
{
  "meta" : {
    "dependent-vtags" : [{
      "resource-id": "network-map",
      "tag": "5feceb66ffc86f38d952786c6d696c79c2dbc239dd↩
          4e91b46729d73a27fb57e9"
    }]
  },
  "endpoint-properties": {
      "ipv4:10.10.2.33" : {
          "network-map.pid": "PID-mov-123",
          "priv:stream-cache": "1"
      },
      "ipv4:10.10.4.12" : {
        "network-map.pid": "PID-mov-321",
        "priv:stream-cache": "1"
      }
  }
}
```

Figure 6.7: ALTO end-point property service used to discover end-points providing movie cache services.

(i.e. congestions, delays). The following section proposes three methods to derive the data-routing cost-value based on the topology and status of the network. The proposals are evaluated according to the following criteria:

- Relative accuracy. The cost metric should reflect the network's topology and load conditions. The metric should also change when the network state changes.

- Calculation complexity. The cost metric should quickly calculate to allow short ALTO server response times.
- Cacheability. The cost metric, once calculated, should be stored in the server's memory to prevent recalculation.
- Interoperability. The cost metric should be calculated in a way that allows a composite metric (metric for data path crossing several networks) to be calculated.

The first proposed metric derives the data-routing cost value from the number of routers along the data-path. This approach follows the routing metric used in the RIP routing protocol [135], albeit without the 15 hops limit. The routing cost metric is increased by 1 for each router that the data passes between the source and destination addresses. This method is referred to as *Hops-Routing-Cost* (HRC).

The second metric derives the cost value from the Interior Gateway Protocol (IGP) used in the ISP's network. Experiments in this work use Open Shortest-Path First (OSPF) as an IGP protocol, and the cost value is calculated by following the OSPF protocol specification [136]. This metric is referred as *Ospf-Routing-Cost* (ORC).

The third metric derives its cost value from the end-to-end available bandwidth along the data-path [137]. The value of the cost metric is equal to the smallest available single link's bandwidth in the links that data traverses along the data-path and is referred to as *Path-Residual-Bandwidth* (PRB). For the first two metrics, a path with a lower metric value is preferred to a path with a higher value. For the PRB metric, the relation is the opposite: a path having a higher value is preferred to a path having a lower value.

Among the three metrics, PRB most accurately reflects the current network conditions, as it is derived directly from the network link load values. The ORC metric is less accurate, as it takes into account only the provisioned link capacities and not the actual load. The HRC is the least accurate, because it indicates only the number of crossed routers.

When an ALTO server is deployed in networks with a simple topology, the ORC metric is the easiest to calculate. It can be done by looking up the value in the first router having a full routing table (routes to all network prefixes) along the data path. In complex networks (networks with multiple OSPF areas), the ALTO server will have to trace the full path between the source and destination addresses. This is due to the fact that routers have a full view of only the OSPF area they are operating in. Deriving the HRC value is more computationally demanding, as it requires the ALTO server to always trace the complete data path from the source address to the destination address. When the ALTO server has routing data from all routers in the network, the HRC value can be calculated in linear time, because the shortest-path is already computed by the routers. Calculating the PRB requires the most resources: in addition to tracing the data path between the source and destination addresses, the ALTO server has to lookup the provisioned link capacities and calculate the average links' load.

Caching of the calculated data-routing cost values is an important consideration in ALTO servers handling a high number of user queries. An operator of an ALTO server

might want to re-use the calculated cost value, when the data path is between the PIDs, for which cost was calculated recently. Once calculated, the ORC and HRC values can be cached for as long as there are no network topology changes and load-based routing is not used. However, the PRB cost value is valid only for a short time. The value changes when the next observation of network links utilization is completed.

The current version of the ALTO protocol does not define the interface between ALTO servers running in different ISP networks [138]. At the same time, it is worth considering, how each of the different cost metrics would interoperate, once the interface is standardized. The HRC metric is the easiest to interoperate with, since it is derived strictly from the network's topology. When a data-path crosses several ISP networks, the composite metric is the sum of individual metrics from each ISP's network. In a similar manner, the PRB composite metric is equal to the minimal value of all metrics in each ISP network. Of the three metrics, the ORC is the least interoperable due to two reasons. First, not all networks use OSPF as their IGP, and other widespread IGPs (e.g., IS-IS, EIGRP) have incompatible cost metrics. Second, the cost value of each OSPF link is derived as a ratio with the reference bandwidth [136]. As different network operators can use different reference bandwidth values, the ORC values from different networks can not be directly compared.

## 6.4 ALTO Server and Experimental Setup

In order to evaluate the proposed methods to derive data-routing cost value, a prototype ALTO server was implemented.[2] The block diagram of the prototype ALTO server is shown in Figure 6.8.

For evaluation purposes, the server maintains test network topology (described later) in the topology database. The topology is represented as a bidirectional graph, where vertices represent network devices and directed edges – links between the devices. In addition to the topology database, the server also maintains information about routing data and links load. This information is maintained in the Network Routing and Load Database. In addition to the routing information snap-shots, load information is stored as a time-series, allowing implementation of complex data analysis.

The server communicates with other network elements through two external interfaces. An infrastructure interface is used to connect to other network devices, such as switches and routers. For evaluation purposes, this interface connected with data collection agents running in the virtual network devices used for testing, using application-specific protocol. However, other protocols, such as Simple Network Management Protocol (SNMP)[139] could also be used.

Handling of ALTO protocol queries is implemented through the ALTO REST interface. Upon receiving a request, the server validates the syntax of the request in the REST

[2]Source-code available online at `github.com/justas-/PyALTO`

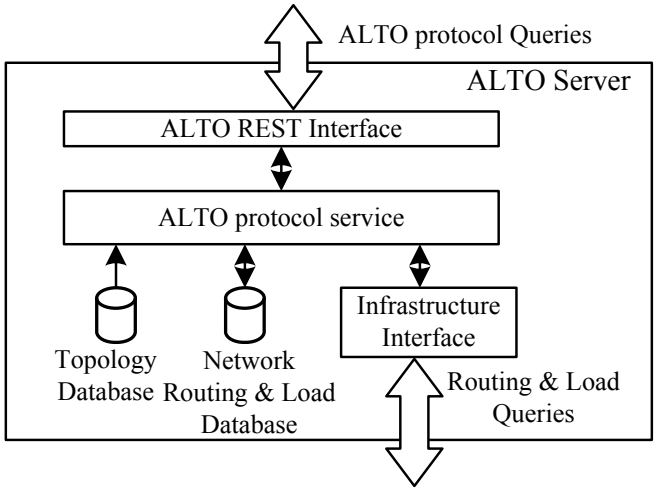



Figure 6.8: Block diagram of ALTO server implementation

interfaces and forwards it to the protocol service for processing. Once the response's computation is done, it follows the reverse path to the client.

Tests to evaluate the different cost metrics were carried out in an emulated environment, using the CORE network emulator. During all tests, users performed a P2P-assisted download of a 5 minute video (HD720p, 2 Mbps codec bit rate) originating from the multimedia server.

The test network topology was adapted from an industry's whitepaper [123] and is shown in Figure 6.9. The test network contains 96 user-nodes divided into 16 groups (PIDs) of 6 nodes. In the tests, user-nodes are connected to the network using Digital Subscriber Line (DSL) technology. This is done by connecting each group of users to a Digital Subscriber Line Access Multiplexer (DSLAM), which in turn is connected to a Broadband Network Gateway (BNG). Every BNG is connected to two DSLAMS, and all BNGs are connected in a ring topology. Data flows from users are always routed via BNGs, and DSLAMs act purely as layer-2 data concentrators. Links between the users and DSLAMs were provisioned with 10 Mbps capacity, and links between the DSLAMs and BNGs were provisioned with 30 Mbps capacity, giving the DSLAMs an oversubscription ratio of 1:2. This allows the experiments to emulate the congestion conditions in the access network. The remaining links were provisioned with 100 Mbps capacity.

Two BNGs have direct connections to the core router and background-traffic generating servers. The core router is connected to an additional traffic generating server, the ALTO server and a multimedia streaming server. The traffic servers are used to generate background data flows to the user-nodes in the network, and the multimedia server acts as

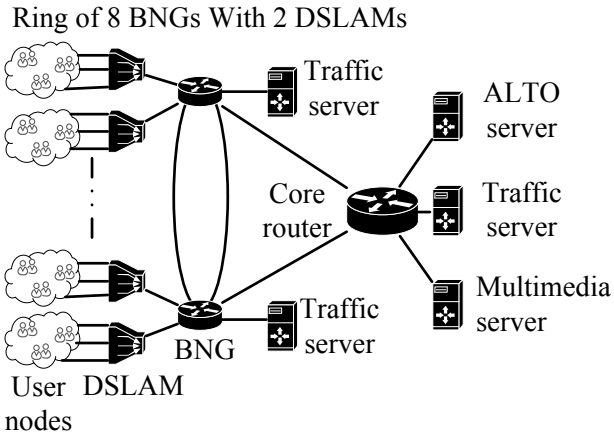



Figure 6.9: Topology of the test network

a source of multimedia data.

The PPSPP and ALTO specifications do not define how the ALTO-provided cost-metric should be integrated into the P2P data download scheduling algorithm. Due to this, the algorithm shown in Figure 6.10 is proposed. The algorithm works as follows: in the experiments, PPSPP clients maintain a list of objects, representing other users participating in the P2P data exchange. Once every second, a timer running in the PPSPP software client calls "request_data" function. The function takes a list of peers, ordered by the ALTO cost-metric, as a parameter. This function tries to request up to 500 data pieces from each member, while ignoring those members that have more than 350 data pieces requests already outstanding.

### 6.4.1 Test Scenarios

This work considers *Video-on-Demand* (VoD) and *Live* usage scenarios in varying network conditions. In the VoD scenarios, the multimedia server has all multimedia data available at the start of each experiment; in the Live scenarios, the multimedia server is producing multimedia data during the experiment. While the scenarios are identical from the point of view of the client receiving multimedia (experiments do not consider fast-forwarding multimedia), the scenarios differ in how much data each client can buffer before rendering it. In the Live scenarios, clients can request only the data that the multimedia streaming server produced. In VoD scenarios, a client always try to maintain a received-data buffer of 1500 data chunks.

To make test conditions realistic and to vary the network utilization, the experiments were conducted using different background data traffic levels. In the base scenario

```
function REQUEST_DATA(alto_ordered_peers_list)
    for peer in local_peers_list do
        set_all_outstanding += peer.outstanding_chunks
    end for
    for peer in alto_ordered_peers_list do
        if len(peer.set_outstanding) > 350 then
            continue
        end if
        chunk_candidates = peer.set_have
        - set_i_have
        - set_all_outstanding
        if len(chunk_candidates) > 500 then
            peer.request_data(chunk_candidates[0:500])
            set_all_outstanding += chunk_candidates[0:500]
        else
            peer.request_data(chunk_candidates)
            set_all_outstanding += chunk_candidates
        end if
    end for
end function
```

Figure 6.10: Data chunks selection algorithm integrating ALTO cost metrics

("traffic-0"), no background data traffic was used. The normal data-traffic level ("traffic-1") was obtained from recent research [124], indicating that 95% of home-users transfer up to 64.27 MB of data during their 15 min peak usage interval. The share between the download and upload traffic was 85% to 15% [125]. In the high background traffic scenarios ("traffic-2"), background data traffic levels were doubled.

The experiments described here were performed using conventional TCP and not LEDBAT as a congestion control mechanism. Due to this reason, the threshold values in the above algorithm are different from those discussed in the earlier chapters.

All experiments were conducted by sweeping the number of P2P multimedia streaming users from 10 to 50 with a step of 10 users. To increase the statistical validity, all experiments were repeated 5 times and results averaged. A set of experiments not using ALTO, and performing random peers selection, were used as a reference case when evaluating ALTO.

### 6.4.2 Evaluation Metrics

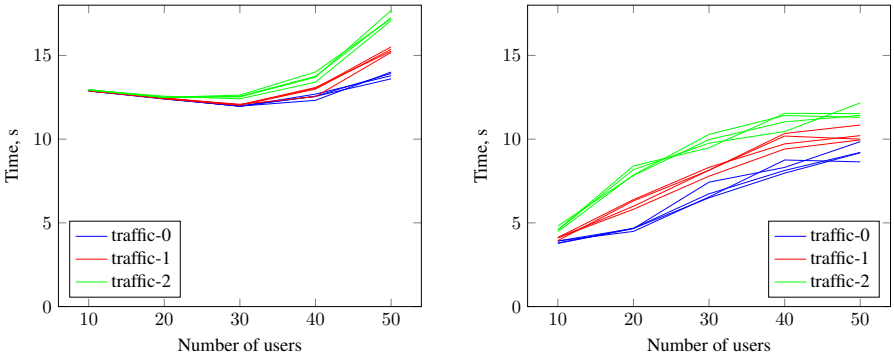
As with other experiments described in the thesis, the two main observed parameters were the playback continuity index and start-up time. In addition, the experiments using ALTO also considers data traffic localization and the multimedia server's load reduction.

In previous works using the AS-path length as a cost metric [131]–[134], one of the goals of using ALTO was to localize the data traffic to the originating AS. This work considers data traffic localization to the BNG connecting a user to the rest of the network. It is assumed that the marginal cost of the DSLAM-BNG link is 0. It means that the cost of using a link does not increase with the increase of data traffic (e.g. a link is a dedicated connection or a virtual connection with committed data rate). Hence, the ISPs have an incentive to keep data-traffic localized to BNGs and not propagate it further in the metro or core network.

## 6.5 Experimental Results

### 6.5.1 Start-up Time

The comparison is begun by measuring the average start-up time of P2P clients when using different cost metrics. The observed average start-up time based on the number of users and background data-traffic levels is shown in Figure 6.11. In both use cases, lines indicate the average start-up time using each cost-metric at the indicated traffic level.




(a) Live use-case

(b) VoD use-case

Figure 6.11: Average start-up time in Live and VoD use-cases with different background data traffic levels.

Figure 6.11 shows that the start-up time increases together with the increasing number of users and background traffic levels. However, start-up time is not affected by the cost-metric, as lines representing each individual metric are all clustered together. The reason for this is two-fold. First, peers in P2P systems make requests to download data based on the data availability. Only if several peers have the same data, can then they be ranked with the help of ALTO. In this case, as soon as the multimedia server indicates that it has new data available, all users request this data, regardless of the cost metric assigned to the source node. The second cause of the indifference in the cost metric is

the design of the experiments. In the experiments, all nodes running P2P software started at the same time (flash-crowd arrival). There are two reasons for choosing this method of node arrival. First, it is the hardest use-case for P2P streaming systems and hence the results show worst-case scenarios. Second, such a nodes arrival pattern is experienced by content delivery networks during high demand periods, such as beginning of sport events or release of popular TV-series. In future work, it is planed to perform tests with both constant and varying nodes arrival rates. This way, a node arriving when other nodes are already functioning, would be able to use the ALTO cost metric to select the best peer to get the multimedia data from.

### 6.5.2 Playback Continuity

The impact on the playback continuity of using different cost metrics, and the reference "No ALTO" case is analyzed next. The average observed playback continuity results for each method of deriving the cost metric are show in Figure 6.12

The comparison begins with the VoD use-case. In the reference "No ALTO" case, the playback index gradually decreases, as the number of users in the network increases. The HRC and ORC metrics perform very similarly, with the playback continuity index above 0.65 for all numbers of users. The PRB performs worse, but still better than the reference "No ALTO" case. It is important to note that the HRC and ORC metrics perform equally well in all background traffic levels, while the performance of PRB method decreases, as the background data-traffic levels increase.

Continuing with the Live use case, it can be seen from Figure 6.12, that the different cost metrics have little impact on the playback continuity index. As explained in the previous subsection, nodes in the "Live" use-cases implicitly select communication peers first based on the availability of data and not on the connection parameters to the peer.

### 6.5.3 Data Traffic Localization and Load Reduction

Since deploying ALTO requires the cooperation of the network operator, this section considers how it impacts data transmissions localization and streaming server load reduction. Here localization means influencing users to exchange P2P data with other users connected to the same BNG. Localization of data transmissions is important to ISPs, because it reduces the amount of data that traverses the operator network. Figure 6.13 shows the average number of data messages received by each P2P user based on the origin of data pieces, and usage scenario for each cost metric. Here, "Local" refers to the data pieces received from the users connected to the same BNG, and "Remote" – to data pieces received from users connected to other BNGs and not the streaming server. Figure 6.14 shows that in both Live and VoD scenarios, using the ORC and HRC metrics increases the number of data pieces received from the users connected to the same BNG. However, the PRB metric is not as effective at localizing data traffic. Comparing the VoD (Figure 6.13a) and Live (Figure 6.13b) use-cases, it is important to note the different vertical scales. Taking the HRC metric as an example: each user received on average 373 data pieces

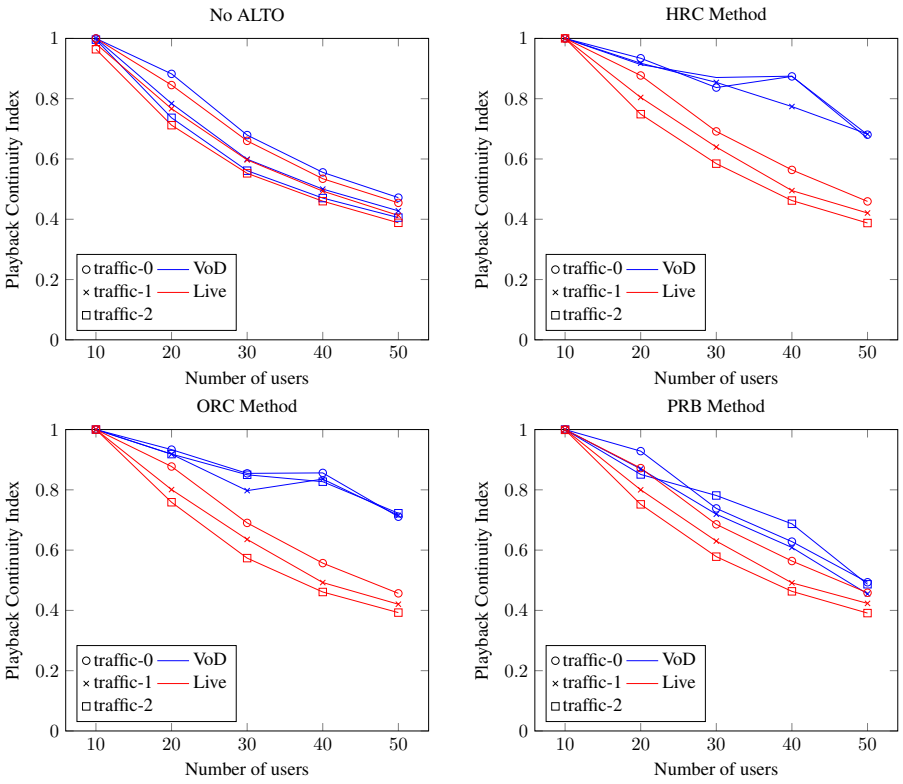
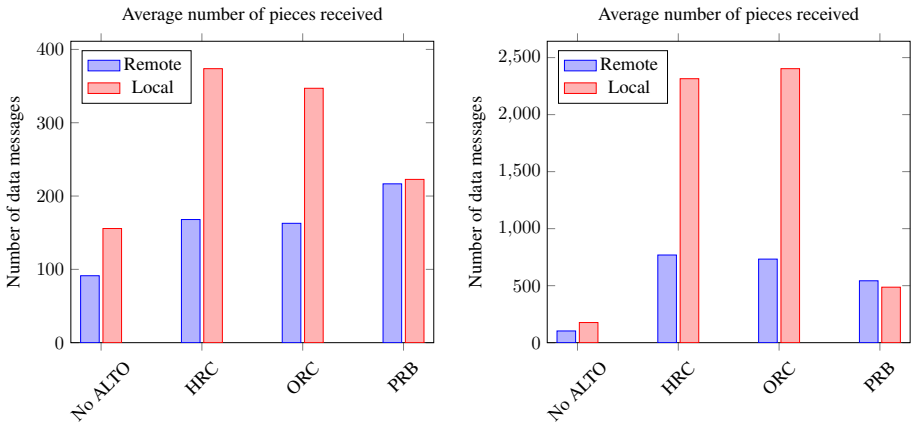



Figure 6.12: Playback Continuity Index (PCI) for each cost-metric based on number of users, usage scenario and background traffic levels.

from other users connected to the same BNG in Live use case, compared to 2314 data pieces in VoD use case. This further reinforces an argument, that in Live use-case, peers receive most of the data from the streaming server, reducing the utility of ALTO.

The final part of the evaluation considers how ALTO can reduce the load on streaming servers. Reducing the load on the streaming server allows the ISP to serve more users with the same server infrastructure. If each user makes less data requests to the multimedia server, then a higher number of users can be served with the same server resources. Figure 6.14 shows the average number of data pieces received from the streaming server, when using different metrics, in both use-cases. The biggest reduction of streaming server originated messages, in the Live and VoD use-cases, is observed when users use the ORC metric, closely followed by the HRC metric.




(a) Live use-case

(b) VoD use-case

Figure 6.13: Average number of data messages received by users based on the communications peer location, use-case and cost metric.

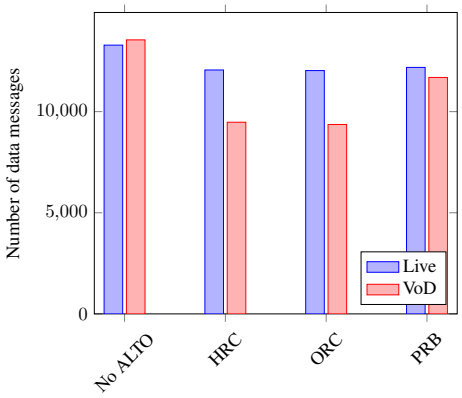



Figure 6.14: Average number of messages received from the streaming server.

## 6.6 Summary

This chapter explores the possibility of using external services to rank the communication peers in P2P communication systems. The primary goal of such ranking is to influence peers selection, so that multimedia is exchanged with peers having favorable connection conditions, which in turn ensures higher quality multimedia streaming.

The presented analysis of the works already in the field indicate that there are two

main methods to evaluate network conditions. The first group of proposals works by assigning coordinates to the network node and then estimating the conditions of the connection using the distance between the coordinates. The second group of proposals uses centralized service to estimate the conditions of the connection.

This work proposes to use the ALTO server to rank the peers by estimating the abstract data routing cost between them. This work also proposes three methods that can be used to calculate the data routing cost between the network endpoints. The proposed methods take into account network topology (HRC method), topology and provisioned link parameters (ORC), or current link loads (PRB) when calculating data routing cost.

The P2P-assisted multimedia streaming tests indicate that both "Hops Routing Cost" and "OSPF Routing Cost" methods increase the QoS of multimedia in the test network. In addition to increasing the QoS, the results indicate that the data exchange between clients can be localized. This is beneficial to the network operators, because it reduces the amount of data traversing the network, thus potentially reducing the network operation costs.

While the results indicate that ALTO can provide benefits to users and network operators, it is not widely used. The reasons for this are varied. First, using ALTO requires the network operator to make information about conditions of its network to be public, which might not be in the business interests. Second, the current version of ALTO protocols does not define the operation of ALTO when data traverses several networks, thus limiting the use of ALTO in the Internet. However, as the following chapter will show, ALTO can be used in single provider networks. The presented use-case shows how ALTO can be utilized to discover cache servers located in mobile base stations.

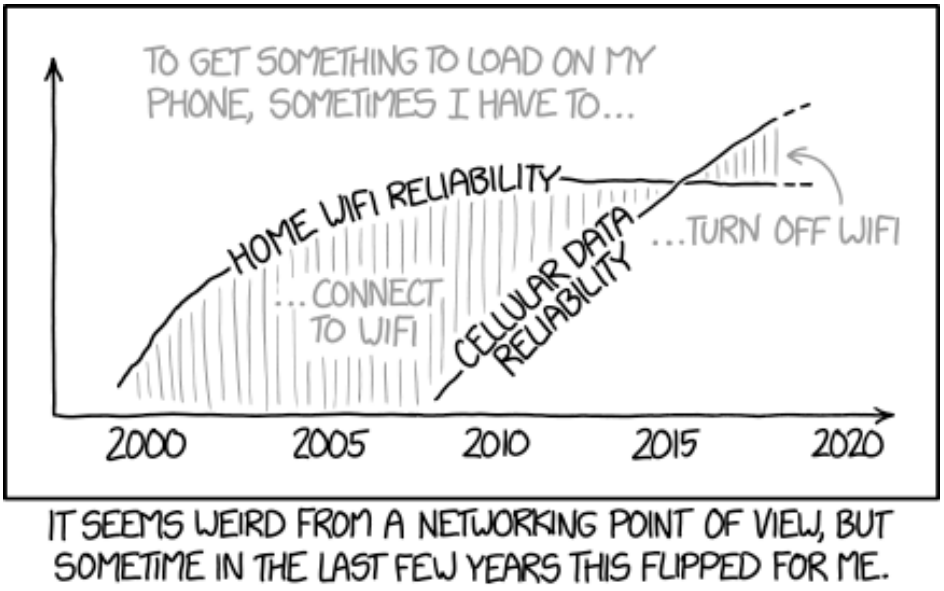



Figure 6.15: A subjective take on two technologies discussed next by a NASA-engineer-turn-cartoonist. © CC-BY-NC, Randall Munroe, `xkcd.com/1865/`

CHAPTER 7

# P2P Communication Over Wireless Connections

All the experiments described in this thesis until this point considered P2P clients connected to the network over the wired Ethernet connection. However, mobile phones are becoming an indispensable part of our life. Recent research indicates that smartphone penetration (number of users in population) is between 60 and 80% in western Europe[140]. At the same time data usage of devices is growing at a high rate. According to an industry report [140], data usage by smartphones is growing at 42% Compounded Annual Growth Rate (CAGR). Of all data transferred to mobile devices, the amount of video and audio streaming data is growing at 50% and 34% CAGR, respectively. Furthermore, between 30% and 40% of smartphone users stream multimedia data during their commute[140]. The widespread usage of smartphones to receive streaming multimedia allows the deployment of new services, such as location-based information, entertainment, and advertising.

The following two chapters of the thesis will explore the proposal of delivering streaming multimedia to mobile devices over wireless technologies. This chapter considers Wi-Fi Peer-to-Peer as a communication technology and the next chapter will consider Long Term Evolution mobile networks as data carriers for P2P communication.

Wi-Fi Peer-to-Peer is a rather recent extension to the widespread IEEE802.11 Wi-Fi standard. As described later in Section 7.1, Wi-Fi P2P can be used to establish a direct connections between user devices without using an Access Point. Furthermore, in contrast to an ad-hoc Wi-Fi mode, devices supporting Wi-Fi P2P communication can maintain connections to the Wi-Fi Access Point and Wi-Fi P2P devices concurrently. Having a possibility to maintain both conventional and P2P connectivity, Wi-Fi P2P can be used to provide new kinds of services. An example of such service, described in Section 7.8, is multimedia streaming in urban trains.

As in this whole thesis, the protocol of choice for multimedia streaming is PPSPP. However, due to specific implementation details of Wi-Fi P2P, PPSPP protocol had to be extended to support operation over Wi-Fi P2P connections. The challenges of using PPSPP and the proposed solutions are described in Section 7.3.

Finally, in order to evaluate the proposed changes, experiments were performed using both real devices and emulated networks. Section 7.10-7.11 presents the results of this

experimental evaluation.[1]

## 7.1 Wi-Fi Peer-to-Peer

In the conventional, infrastructure mode of IEEE 802.11 Wi-Fi, devices must connect to an AP to communicate with each other. An AP is typically a dedicated hardware capable of performing functions related to security, power-saving and QoS in its network (called a Basic Service Set (BSS)), and connects the network to a wired network, such as the Internet.

Today, users like to interact with each other on their devices on a frequent basis, without the need for first connecting to an AP. Thus, a sharp rise in the popularity of device-to-device (D2D) or Peer-to-Peer communication technologies is currently being witnessed. Although 802.11 Wi-Fi does support D2D communication with its infrastructure-less mode, also known as "ad-hoc mode", the feature has witnessed limited popularity owing to it being rather user-unfriendly[141]. In ad-hoc mode, devices can communicate with each other directly, without the involvement of an AP, forming an Independent Basic Services Set (IBSS).

To enable more efficient means of D2D communication for Wi-Fi users, the Wi-Fi Forum adopted an extension to the IEEE 802.11 standard called Wi-Fi Peer-to-Peer (Wi-Fi P2P, also known as Wi-Fi Direct) [142]. It is worth noting that Wi-Fi P2P is not an IEEE standard itself. Rather, it is a technical specification that defines the architecture and protocols to implement Wi-Fi based D2D communication. In Wi-Fi P2P, devices communicate in groups. In a group, a Wi-Fi P2P device may have the role of the Group Owner (GO) or of a client. The GO role is negotiated at the time of the group formation with the help of an election process. A GO is also occasionally referred to as a "Soft AP". It provides group clients with AP-like functionalities. The client devices may include both legacy (i.e. conventional 802.11 devices) and P2P clients [141].

As in a conventional Wi-Fi network, a P2P group has an SSID. This SSID always starts with ASCII characters "Direct-" to distinguish itself from conventional Wi-Fi SSIDs. Like a conventional AP, a P2P GO advertises the group by broadcasting this pseudo-random SSID [142]. A GO is responsible for assigning IP addresses to the clients as well.

A Wi-Fi device must support IEEE 802.11g or a later technology to act as P2P device [142]. In contrast to a conventional infrastructure or ad-hoc Wi-Fi network, a P2P device can act as a GO and be a client in other groups simultaneously, provided that it supports multiple wireless interfaces or can implement virtual interfaces or time-sharing techniques on the same interface [141], [143]. Since a GO is responsible for assigning IP addresses in the group, and there is no coordination among GOs, different groups can end up using the same address range. The Wi-Fi Direct standard does not specify a method for resolving these addressing conflicts.

[1] Contents of this chapter is based on the papers 1, 4, 6 from the thesis publication list.

Legacy Wi-Fi devices can also communicate to a GO—they do not formally belong to the group but simply "see" the GO as a conventional AP [142]. Note that the P2P functionality is realized by the introduction of P2P IEs (Information Element) in the management frames. Legacy devices ignore these IEs [142].

A P2P device can both be a part of a group and maintain a conventional Wi-Fi connection simultaneously, as shown in Figure 7.1. Such a device is referred to as a P2P Concurrent Device [141]–[143]. This implies that, for example, in a P2P group, a device can provide the group clients with a connection to the outer world by simultaneously connecting to a conventional Wi-Fi AP or 3G/4G network. This, again, is a major advantage over a conventional ad-hoc network in which devices cannot simultaneously connect to existing Wi-Fi networks [142].

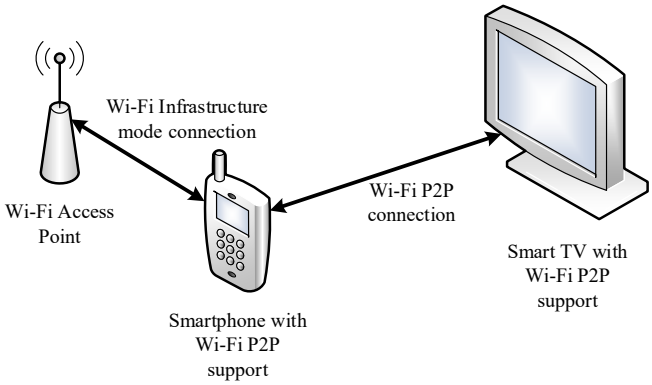



Figure 7.1: A smartphone with Wi-Fi P2P support can connect to the Wi-Fi Access-Point and another Wi-Fi P2P device simultaneously.

Thus, in contrast to a conventional Wi-Fi network, in which a device can either operate as an AP or a client, in Wi-Fi P2P these roles are arbitrary—any device can function as an AP. Furthermore, given that this new capability can be implemented entirely in software, without the need for making any changes to the hardware, bears a high significance [143]. For instance, in a conventional Wi-Fi network, the power-saving capability is not defined for APs but only for clients, as an AP is often a device with a superior set of capabilities. In contrast, a GO, being any ordinary device, might have limited battery power and thus might need the power-saving capability. Since Wi-Fi P2P is built upon the conventional infrastructure Wi-Fi, a P2P device automatically inherits all of these features [143].

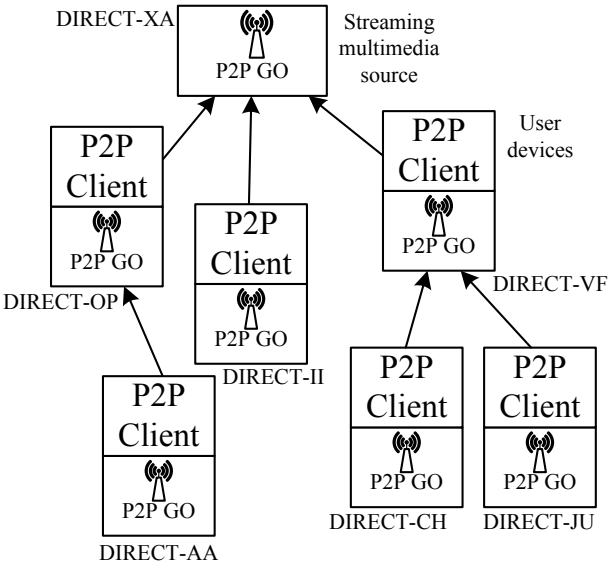



Figure 7.2: Communication model of the multimedia streaming system. Each user device performs functions of Wi-Fi P2P GO and client devices. Arrows indicate P2P member-of relationship.

## 7.2 Adapting PPSPP for streaming over Wi-Fi P2P

In order to enable multimedia streaming over a network formed using Wi-Fi P2P connections, two tasks must be performed concurrently. First, the Wi-Fi P2P connections must be established between the devices. Second, once the connections are established, PPSPP must establish connectivity between the devices and exchange information about data chunks available to each client. As described previously, each Wi-Fi P2P device can maintain multiple concurrent connections at the same time. Using this property of Wi-Fi P2P allows to create a "tree-like" network, as shown in Figure 7.2. Here, six user devices are receiving multimedia from the streaming multimedia source. Each user device is working as the GO and as a client at the same time.

The protocol stack of three clients performing multimedia streaming is shown in Figure 7.3. In the figure, a streaming server is working as a GO and has a PPSPP client binded on the Wi-Fi adapter working in the GO mode. In mobile devices #1 and #2, there are two virtual Wi-Fi adapters, one working in GO mode and another in client mode. Both clients run an instance of PPSPP software, which is binded to both virtual Wi-Fi adapters. As the figure shows, each non-root (non streaming-server) is maintaining at

least one PPSPP connection – to the PPSPP client running in the GO device. At the same time, each client acts as a GO itself, and maintains PPSPP connections to all other clients connected to them.

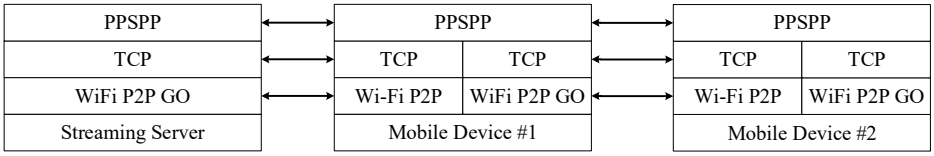



Figure 7.3: The protocol stack of three devices performing multimedia streaming over Wi-Fi P2P connections

The described tree-like setup, when used with PPSPP (or any other P2P communication protocol), allows to distribute multimedia data to all interconnected clients without using any data Open Systems Interconnection (OSI) network-layer routing protocol. The process of data distribution to the connected client is shown in Figure 7.4. The first data exchange (1) in the figure shows the multimedia server informing client #1 about the availability of multimedia data using the HAVE messages. The client then requests the data using the REQUEST messages (2). The server replies with the data messages to the connected client (3).

The same process is then repeated one level down the tree. The first client informs clients #2 and #3 about the just-obtained data using HAVE messages (4). The remaining clients then request the data using REQUEST messages (5). The requests are fulfilled using DATA messages (6). By repeating this process, eventually all connected clients receive all data advertised by the multimedia server.

## 7.3 PPSPP Protocol Changes

The PPSPP protocol, as defined in [80], is meant to be used in IP networks where all users can establish connections to other users. This is not the case when used with Wi-Fi P2P. Users in P2P groups can communicate only with the GO and (if GO settings allow) other members of the same group. In order to make PPSPP work over the networks formed using Wi-Fi P2P connections, a number of changes to the protocol were implemented, as is described next.

The first change is related to the way new connections are made to other users. In conventional PPSPP, each new user learns about other users and their IP addresses from the tracker server or other users. This approach works well in conventional networks, where most users can make connections to other users. In the Wi-Fi P2P communication model, users of one group can communicate with the GO and other users in the same group. To enable data exchange between the different groups, each user must establish a

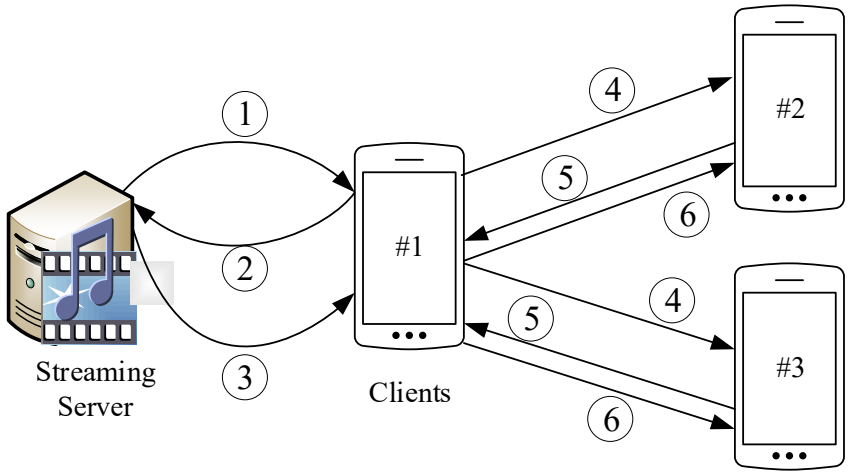



Figure 7.4: The process of data distribution from the multimedia source to all connected clients.

connection to another Wi-Fi P2P group. To do so, users send to the tracker server the SSID name and the MAC address of their Wi-Fi adapter that is working as the Access-Point (AP). When a new user joins, it receives a list of all active SSIDs from the tracker server. Then it joins one of the randomly selected group.

The second change proposed to the PPSPP protocol modifies the process of exchanging information about other users via the Peer Exchange mechanism. In the conventional PPSPP, each user can periodically try to learn about other users using PEX_REQ and PEX_RES messages. However, as the user's IP addresses is valid only in its group, the PEX_RES message was extended to include the SSID names and MAC addresses as well. These extended PEX_RES allowed users to establish connections to other Wi-Fi P2P groups.

The third enhancement to the protocol is related to the unique identification of each user. In the conventional PPSPP, each user is uniquely identified by the data tuple containing the IP address of the user and the port number the PPSPP client is listening for data. This prevents two users from making concurrent connections to each other. However, consider a situation shown in Figure 7.5. Here, two clients connect to each other's Wi-Fi P2P group. As communication over Wi-Fi P2P groups involves communication concurrently over several virtual adapters (each with its own IP address), a different method is needed to uniquely identify each user and prevent such a situation. This is done by creating a type-4 Universally Unique Identifier (UUID)[144]. This UUID value is then added to the HANDSHAKE message that is used to establish a connection between the

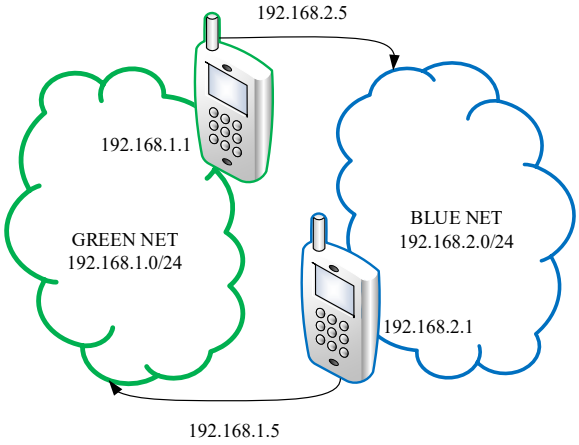



Figure 7.5: Duplicate concurrent connections between two hand-held devices. Each device is a GO and a client at the same time. Arrows indicate member-of relationship.

users. If during a handshake process a user detects that it already has a connection to another user with the same UUID value, it starts a process to decide which connection should be terminated. To avoid a situation where both users close connections at the same time, a strategy was chosen to decide which user should disconnect: both users compare their UUID values, and the user having a lower UUID value disconnects from the group of a user with the higher value.

## 7.4 Feasibility study

Before testing multimedia streaming over Wi-Fi P2P connections using an emulated network, initial tests were performed using real devices to test if groups can be created programmatically. These tests were also used to measure the impact on throughput performance, when data sent from one client to another is relayed in GO.

According to the Wi-Fi P2P technical specification[142], the GO in a P2P group "May provide communication between associated Clients."[142] This indicates that the data relay between the clients in the GO is optional and not a mandatory function. In order to evaluate what impact data relay has on the data transfer speeds, a number of tests were performed using the scenarios shown in Figure 7.6. In the tests, data was sent between the clients and the GO (C2G, CC2G, G2CC cases) or from one client to another, with data relay happening in the GO (C2C case). In all cases, tests were performed using the iPerf3 [121] software using one-way data transfer. This method was chosen to

model multimedia data exchange between the users, where data is disseminated from the multimedia source (and not always the GO) towards the rest of the users. The devices used in tests were Raspberry Pi 3 model B, using the on board IEEE802.11g Wi-Fi adapter. The devices ran Linux 4.9.8 and version 2.6 of wpa_supplicant (Wi-Fi management software). Each scenario was repeated 10 times and the transfer speeds were averaged.

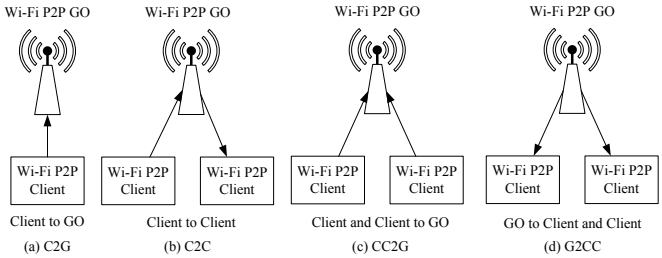



Figure 7.6: Baseline evaluation scenarios. Arrows indicate data transfer direction.

The test results are shown in Figure 7.7, error bars in the figure indicate one standard deviation. Here, it is important to note two outcomes. First, the speed difference between the CC2G and G2CC scenarios is small (4%). In both cases, the average data transfer speed is approximately 18 Mbps. This shows that there is only a small speed difference between sending data to and from the GO. Second, relaying data in GO incurs a significant performance penalty. The average data transfer speed in the C2C scenario where relay is being performed is 13.58 Mbps. This is about 25% slower than in CC2G and G2CC scenarios, where the GO is also serving two data streams, however, the relaying is not

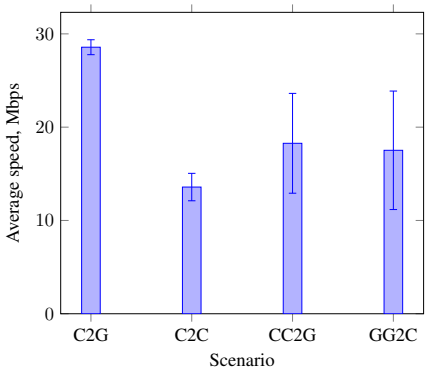



Figure 7.7: Average observed data transfer speeds in base-line testing.

performed.

Based on these observations, the communication model and the changes to the PPSPP protocol were adjusted. In the communication model, communication within a group was allowed only between the group member and the GO. The changes to the PPSPP protocol included changes to the Peer Exchange mechanism as described in the previous section.

## 7.5 Experimental Setup

Once the feasibility study was completed, the remaining evaluation of multimedia streaming performance using Wi-Fi P2P connectivity was performed in a virtual environment using the CORE network emulator [122]. The physical and media access layers of Wi-Fi connections were emulated using EMANE, a part of CORE. The Wi-Fi MAC layer used the IEEE 802.11g standard with the maximum data rate set to 54 Mbps. The experiments used the dipole antenna model and free-space radio waves propagation model.

The experiments considered two types of streaming multimedia. In the *Live* scenarios, one of the users acted as the source of multimedia, injecting new data chunks as they are produced. In the *Video-on-Demand (VoD)* scenarios, one of the users made the whole multimedia record available for download. In all scenarios, the multimedia recording is a 1280x720 (HD 720p) video clip encoded with a VP8 codec having a bit-rate of 2 Mbps.

During the simulations, each client periodically (every second) ran the algorithm shown in Figure 7.8 to request chunks from other clients. Each experiment varied one of three parameters: initial buffer size, forward download window size and number of users performing multimedia streaming. The initial buffer size indicates how many data chunks must be received before the rendering of multimedia starts. The size of the initial buffer was swept between 50 and 150 frames with the step of 50 and the forward download window was swept between 500 and 2000 data chunks with the step of 500. Additionally, a set of experiments was carried out where the forward download window size was not artificially limited. This implied that the users could download as much data as available. Experiments with each combination of parameters were repeated five times and the results averaged.

The experiments described here were performed using conventional TCP and not LEDBAT as a congestion control mechanism. Due to this reason, the threshold values in the above algorithm are different from those discussed in the earlier chapters.

In all experiment's results described next, the clients arrived in the flash-arrival pattern (all at once). This was done intentionally, in order to test the system using the works-case scenario. A different (and more realistic) users arrival pattern is used in the section describing in-train streaming experiments.

```
function REQUEST_DATA
    for peer in local_peers_list do
        set_all_outstanding += peer.outstanding_chunks
    end for
    for peer in local_peers_list do
        if len(peer.set_outstanding) > 350 then
            continue
        end if
        chunk_candidates = peer.set_have
        - set_i_have
        - set_all_outstanding
        if len(chunk_candidates) > 500 then
            peer.request_data(chunk_candidates[0:500])
            set_all_outstanding += chunk_candidates[0:500]
        else
            peer.request_data(chunk_candidates)
            set_all_outstanding += chunk_candidates
        end if
    end for
end function
```

Figure 7.8: Data chunks selection algorithm

## 7.6 Evaluation Results – Start-up Time

The average observed client start-up time for both Live and VoD use-cases is shown in Figure 7.9. The figure shows the average start-up time for Live and VoD use-cases based on the initial buffer size. Note that the tests were carried out by increasing the number of users until the Playback Continuity Index value dropped below the threshold. Hence, in the Live use-case, the largest tested user group contained 18 users. Figure 7.10a shows the cumulative distribution function (CDF) of client start-up times in Live, and Figure 7.10b in VoD use-cases.

Figure 7.9 shows that the start-up time depends on both the size of the initial buffer and the number of users. The size of the buffer has the biggest impact on the start-up time in the Live use-case. This is caused by the fact, that the buffer can only be filled with data that is being produced by the source peer during filling of the buffer. In the VoD use-case, all multimedia data is available immediately, and so the buffer size has a much smaller impact on the fill rate.

The decrease of the average start-up time as the number of users increases in the Live use-case is caused by the communication load sharing between the users. However, this is not the case in the VoD use-case. Here, the average start-up time increases together with

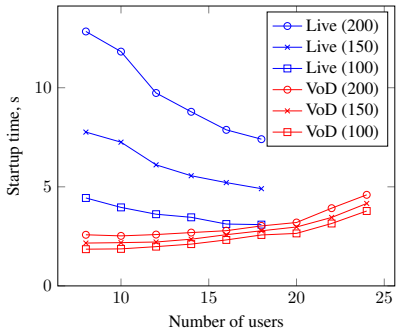
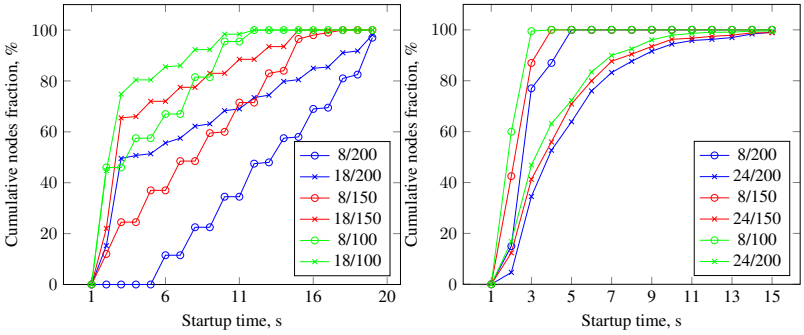



Figure 7.9: Average VoD and Live start-up time

the number of users. The cause of this is a combination of the availability of the whole multimedia file upon experiment start and the use of wireless communication medium. As all users try to fill their playback buffer, they increase the load on the source node in addition to acting as a source of interference to other users.




(a) Live (b) VoD

Figure 7.10: Distribution of the average node start-up time based on multimedia type, initial buffer, forward download window sizes and number of users. Legend values show number of users / window size.

Figure 7.10 shows how the average start-up times are distributed among the users for different buffer and user group sizes. The CDF for the Live use-case shows that when the number of users is large (18), more than half of the users start rendering the multimedia content within the first 3 seconds regardless of the buffer size. However, when the number of users is small and the buffer size is large, the first users start rendering multimedia only after 6 seconds. In the VoD use-case, all users started rendering multimedia within the

first 5 seconds when the number of users was small. However, the distribution is much more long-tailed when the number of users is high. This indicates that while most of the users start rendering multimedia fast (within 3 seconds), there are users having start-up times above 10 seconds.

## 7.7 Evaluation Results – Playback Continuity

The average observed Playback Continuity Index (PCI) for both Live and VoD use-cases is shown in Figure 7.11 which shows the average PCI for Live and VoD use-cases based on the initial buffer size. Figure 7.12 shows the PCI based on the forward download window size for Live and VoD use-cases respectively. In all figures, the horizontal dashed line indicates the threshold PCI value.

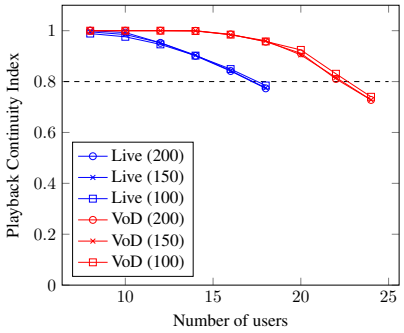
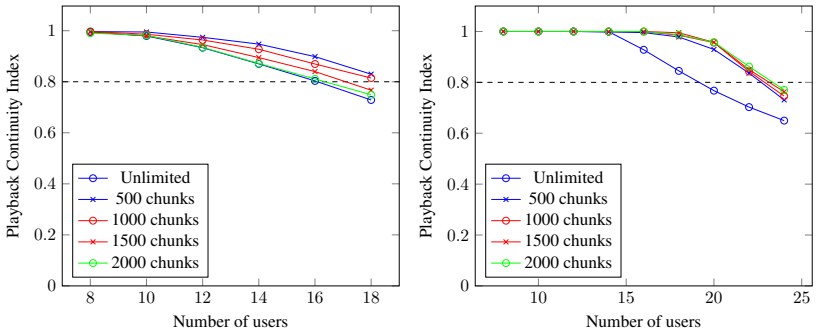



Figure 7.11: PCI in Live and VoD use-cases

It can be seen from Figure 7.11 that the initial buffer size has almost no impact on the playback continuity. The PCI stays at 1 as long as the number of users is no larger than 8 in the Live use-case and 14 in the VoD use case. As the user group size grows, the PCI value gradually decreases to the threshold value for both use-cases.

Figure 7.12a shows the average observed PCI as the function of the forward download window size in the Live use-case. In the corner-case "Unlimited" experiment, the user's forward download window size was not limited, and they could try to download as much data as possible. The figure indicates that for small user-group sizes (up to 10 users), the size of the window does not impact the PCI. However, as the user group size increases, having a smaller forward download window is more beneficial that having a large one. For a group of 18 users, reducing the window size from 2000 data chunks to 500, increases the PCI value from 0.75 to 0.83.

For the VoD use-case, Figure 7.12b shows the impact of the forward download window size on PCI. For up to 14 users, the size of the window has no impact on the PCI value. As the number of concurrent users increases, the PCI becomes lower for users using a




(a) Live (b) VoD

Figure 7.12: Playback Continuity Index based on multimedia type and the forward download window size.

small buffer. When the number of users reaches 24, the PCI changes from 0.73 to 0.77 as the window size increases from 500 to 2000 data chunks. Here it is important to note that users with an unlimited window size experienced much lower PCI. Such users requested all multimedia data immediately at the start, increasing the load on the source node. This, in turn, prevented timely delivery of data to the remaining users.

## 7.8 In-train Multimedia Streaming

Presently, the two most widespread methods to deliver data connectivity to urban rail passengers are (1) by providing a cellular connection directly to each passenger device, or (2) by providing a cellular connection to a gateway device in the train car which is then used to provide connectivity to passengers' devices using Wi-Fi technology, as shown in Figure 7.13. The cellular connection in these two methods can be based on e.g. 3G/4G[145], WiMAX[146], or IEEE 802.11p (WAVE)[147].

The first method, shown in Figure 7.13.A, offers data connectivity directly to a user's device. However, in order for it to work, users (or the train) must be inside the cellular network's coverage area. In addition, interruptions in the data connection can be experienced as a fast moving train roams from one base station to another [148], [149]. Furthermore, since streaming multimedia is a bandwidth-intensive application, it can incur significant data transfer charges due to the usage of cellular connection.

The second connectivity method, shown in Figure 7.13.B, involves providing data connection to a gateway device inside a train, which is then shared among the passengers using conventional Wi-Fi technology. Nonetheless, this method likewise requires cellular network coverage. In addition, providing a secure data connection is problematic when using conventional Wi-Fi. As per the Wi-Fi standard[150], secure communication can

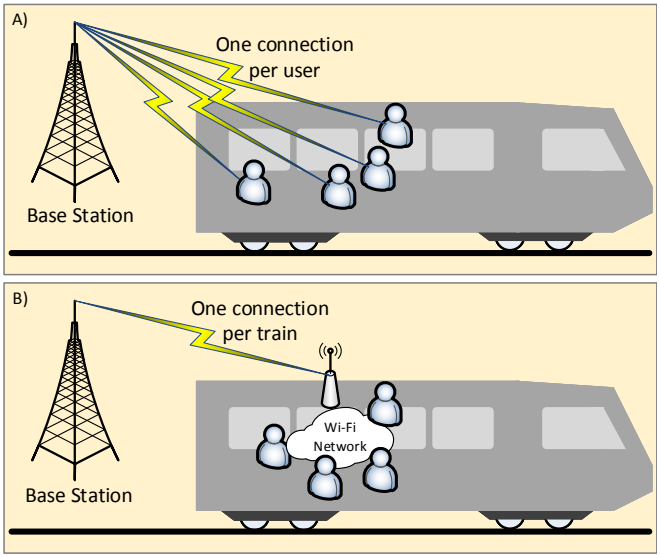



Figure 7.13: Methods of delivering data connectivity to the passengers: (A) using a direct cellular connection for each user, (B) using one cellular connection for the train that is shared with other users using Wi-Fi.

be enabled by encrypting the transmissions either by using a pre-shared key or a user certificate. As these methods are impractical for mass deployment, Wi-Fi connections in public transport are typically unencrypted.

A potential solution in these scenarios is to reduce the downlink connection load. During periods of peak usage, such as sports events or live shows, multiple users will stream the same multimedia content. By doing so, users increase the load on the downlink connection to the train with multiple copies of identical data. By using technologies that reduce the load on the downlink connection, a larger number of users can be served with high-quality multimedia.

In order to address the above challenges, a novel method of streaming multimedia to the train passenger is proposed. The overview of the system is shown in Figure 7.14. In order to distribute multimedia to the train passengers, it is delivered to the train using the available bandwidth of the Communications-Based Train Control (CBTC) connection. The selection of radio technology used to implement the CBTC connection is out of the scope in this work. However, it is assumed that the chosen technology provides enough spare capacity to deliver multimedia data. Once data is delivered to the train, it is distributed among the passengers over the Wi-Fi P2P connections. This is done with the help of the *source node*—a special device acting as a gateway between the CBTC and

Wi-Fi P2P connections and running a client of the P2P multimedia streaming software. Clients use the multimedia streaming software to request data from other clients and coordinate data delivery, using the PPSPP for this purpose.

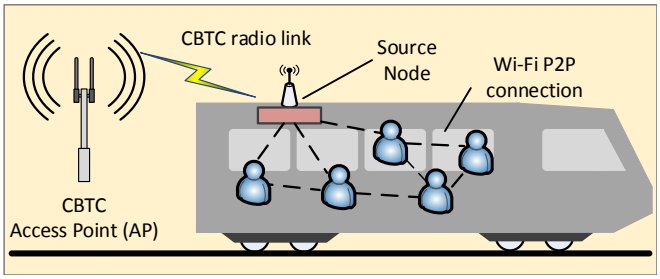



Figure 7.14: An overview of the proposed multimedia streaming system.

The proposed multimedia streaming system addresses all the above listed technical challenges. As a CBTC radio communication network is designed to provide uninterrupted connectivity to the train, there are no areas without radio coverage. Furthermore, there are no data transmission interruptions when using the CBTC system compared to cellular connectivity as described previously.

The security of data communication within the train car is ensured by employing encryption methods defined in the Wi-Fi P2P standard[142]. Furthermore, the Wi-Fi P2P programming interface supports programmatic setup of encrypted communication without user interaction[151]. Finally, using the P2P communication model makes each user act as a proxy server. Once multimedia is delivered to the active user's device, other users can download it from there, instead of consuming bandwidth of the train's CBTC connection. It is important to note that each user is acting as a proxy only while it is streaming multimedia. As soon as the user stops streaming multimedia, he leaves the P2P network to conserve the battery life of the device.

## 7.9 In-train Streaming Experiment Setup

In order to evaluate the quality characteristics of streaming multimedia during user turnover, experiments were designed as described next and shown in Figure 7.15. At the start of each experiment, a number of users started multimedia streaming using the proposed system. The experiments were repeated with 15, 20 and 25 starting users. Then, after a period of time (3, 5 or 7 minutes), either extra users were added (Add scenarios) or part of the users was replaced with new users (Replace scenarios). The number of users added or replaced varied between zero and 80% of the starting population. After adding or replacing users, the experiment continued for the same period of time. Such an experiment setup allowed to measure what impact the adding or replacing of users, at regular time

intervals, has on the streaming multimedia's quality. The scenarios considering removal of users are not included in this work, as the results described next indicate that removing users does not affect the streaming multimedia's quality negatively.

The experiments were performed using VoD multimedia only. The size of the initial buffer was 100 chunks and the size of the forward download windows was 500 chunks.

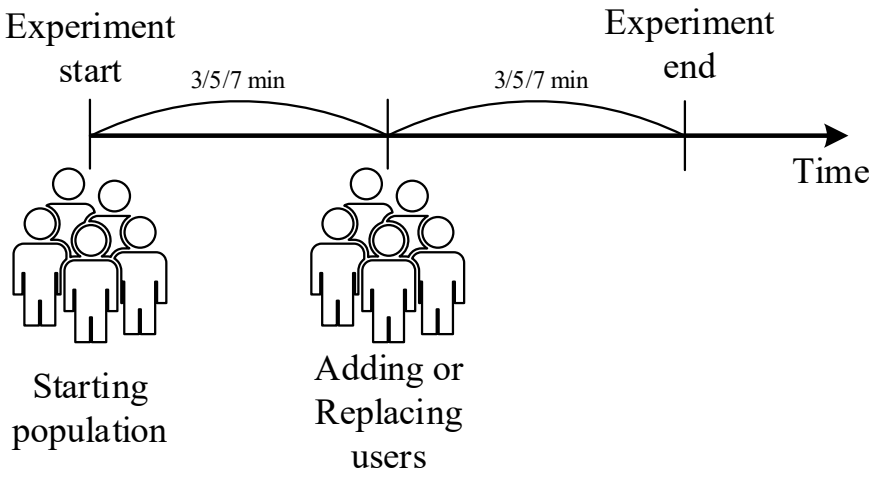


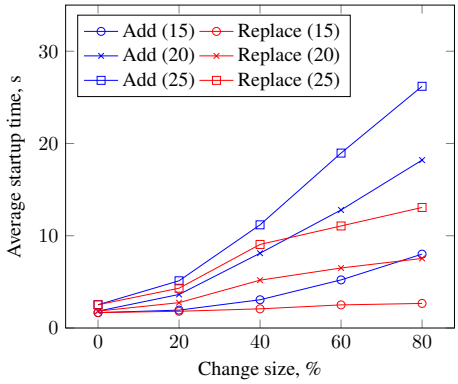

Figure 7.15: Graphical description of in-train streaming experiment setup.

Overall, this work considers 90 different combinations of parameters. Simulations using each unique combination (scenario) of parameters were repeated 5 times and the results averaged. The simulation used the identical setup as previously described in Section 7.5.

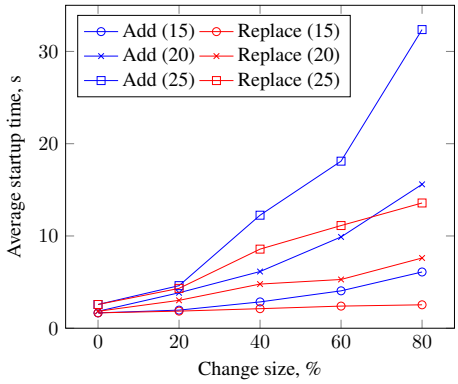
## 7.10 In-train Streaming – Start-up Time

Results for the average client start-up times are shown in Figure 7.16. Here, each sub-figure shows the average start-up times for each of the 3, 5, and 7 minute time intervals before and after user population changes. Comparing the results between the "Add" and "Replace" scenarios, it can be seen that clients in the "Add" scenarios experienced longer start-up times than clients in the "Replace" scenarios. This is especially visible in the 3 minutes case (Figure 7.16a). In this example, when the number of users increases from 20 to 32 (size change of 60%), the multimedia playback started after 6.5 seconds in the "Replace" scenario, and 12.8 seconds in the "Add" scenario.

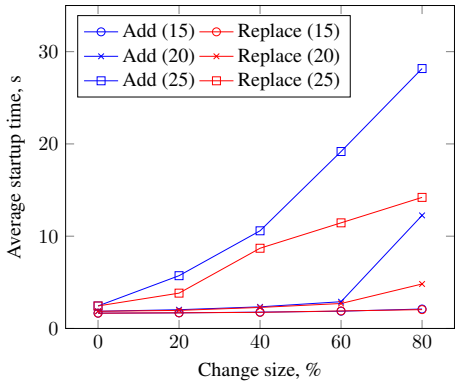
The longer start-up times in the "Add" scenarios are caused by the increase in the number of active users. In the "Replace" scenarios, the total number of active users stays




(a) 3-minute interval




(b) 5-minute interval




(c) 7-minute interval

Figure 7.16: Average observed user start-up times. Numbers in the parentheses in the legends indicate the starting users population.

unchanged (15, 20 or 25 users), while in the "Add" scenarios, the total number of users increases with the change size. As all users communicate using the same Wi-Fi radio channel, a higher number of users introduce more interference, thus reducing the effective transmission speed and increasing the time required to fill the playback buffer.

By comparing the results for the different time intervals, it can be seen that the average start-up time is similar when the interval is 3 and 5 minutes (Figure 7.16a and 7.16b), but decreases significantly when the time interval is 7 minutes (Figure 7.16c). The reduction of the average start-up time here is the result of the chosen video file length of 5 minutes. In a test scenario with the 7 minutes time interval, all users in the system are able to download the file fully before new users arrive. When the new users arrive, they are able to request data from multiple users concurrently, thus reducing the time required to fill the initial playback buffer. For comparison, users in 3 and 5 minutes scenarios are still downloading the multimedia file when the new users arrive.

## 7.11 In-train Streaming - Playback Continuity

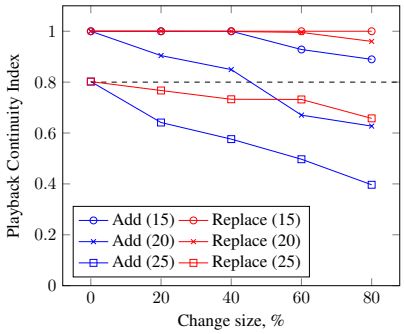
The average observed PCI values for different time periods between the stops are shown in Figure 7.17. For all time periods, the results indicate that the system performs within the chosen quality threshold (PCI $>$ 0.8) when the starting user population is 25 users. This observation is valid for both "Add" and "Replace" scenarios. Furthermore, in all tests using different time intervals, the "Add" scenarios resulted in worse average PCI values than the "Replace" scenarios. As with the start-up time results discussed previously, this is caused by the radio interference created by the users operating on the same Wi-Fi channel, thus reducing the effective data transfer speeds.

Comparing the results for the different time intervals, it can be seen that the average PCI values for the 7 minutes scenario (Figure 7.17c) are higher than those of the 3 minutes scenario (Figure 7.17a). This is especially visible in the "Add" scenarios. For example, when the number of users changes from 20 to 36 (80% increase), the PCI value is 0.62 in the 3 minutes scenario, and 0.83 in the 7 minutes scenario. This indicates that the quality of multimedia is negatively impacted by the frequent change in user population.

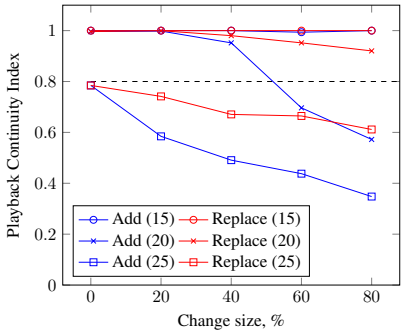
Overall, the obtained results indicate two main outcomes. First, the proposed multimedia streaming system delivers multimedia within the chosen QoS threshold when the number of concurrent users is less than 25. For the urban-trains use-case, this means that each train car should have at least one source node to serve the passengers. Second, the system delivers higher quality multimedia when the time intervals between the stops are longer. This can limit the deployment of the system in railway lines with frequent stops.

## 7.12 Summary

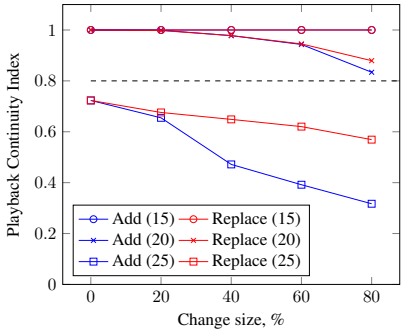
This chapter presents the idea of using Wi-Fi P2P technology to stream multimedia in ah-hoc formed wireless networks. The Wi-Fi P2P technology enables users to establish




(a) 3-minute interval




(b) 5-minute interval




(c) 7-minute interval

Figure 7.17: Average observed Playback Continuity Index (PCI). Numbers in the parentheses in the legends indicate the starting user population. Dashed lines indicate the chosen performance threshold.

multiple direct connections between the technology supporting devices. Such ah-hoc connections can be used to deliver multimedia in new types of services – such as location-based advertising or in-transport multimedia distribution. This chapter also present the changes done to the PPSPP protocol to make it work over the Wi-Fi P2P connections.

Three sets of experiments were also performed to investigate performance parameters of multimedia streaming over the proposed system. The tests performed using real devices indicated that the proposed data transmission model is feasible.

The tests in emulated network investigated how the size of the initial buffer, forward window size and the type of multimedia impacts the start-up times and playback continuity. As with previous results, there is a clear indication, that maintaining smaller buffer and window sizes has a benefit against maintaining larger sizes. Furthermore, the results indicate that in identical systems streaming Live and VoD multimedia, system delivering VoD multimedia allows to reach higher playback continuity to larger number of users.

The final set of tests investigated the idea of using Wi-Fi P2P based multimedia streaming for information dissemination in trains. The results showed that the proposed system is feasible when the number of users is up to 20 and the users turnover is small.

CHAPTER 8

# P2P Communication in the Long Term Evolution Networks

Mobile phones are integral to everyday communication. The amount of data delivered to mobile phones is growing every year and is expected to grow for the foreseeable future. Multimedia data contributes to a large part of this growth. At the same time, the architectures of mobile networks are undergoing constant development. New network architectures are being proposed to satisfy connectivity requirements depicted in the 5G (5th Generation) scenarios. Among the proposed architectures, there is a clear focus on a Heterogeneous Network (HetNet) with diverse types of Base Stations (BSs), the distribution of the content at the edges and the centralization of functionalities [152].

In contrast to the previous chapter, this chapter deals with the mobile network infrastructure and not the P2P communication software. In this chapter, a method of storing multimedia content in the mobile network base stations is proposed. Furthermore, a prototype of the proposed solution was implemented and its performance tested.

The said prototype was used to deliver data stored in the base stations to the mobile clients. Later, the prototype was extended to support infrastructure-assisted Device-to-Device (D2D) communication. Such infrastructure-assisted D2D communication can be used as a building block in the next generation distributed (so called Fog) networks.

In the rest of this chapter, Section 8.1 provides a short primer introduction into the architecture of LTE networks. The architecture of the proposed base station caching solution is described in Section 8.3. It is followed by the experimental performance evaluation. The proposed solution is then extended to support D2D communication as described in Section 8.5. Finally, Section 8.7 provides a summary of the main findings.[1]

## 8.1 LTE Networks Primer

Mobile networks are becoming part of most critical infrastructures for societies. Mobile networks are constantly undergoing evolution to support new services. At present, the most widespread mobile networks technology is LTE, as described by the industry standardizing body called 3rd Generation Partnership Project (3GPP). The main elements of the LTE

[1] Contents of this chapter is based on the paper 2 from the thesis publication list.

network are shown in Figure 8.1. It is important to note that only the elements relevant to the work presented in this chapter are described.

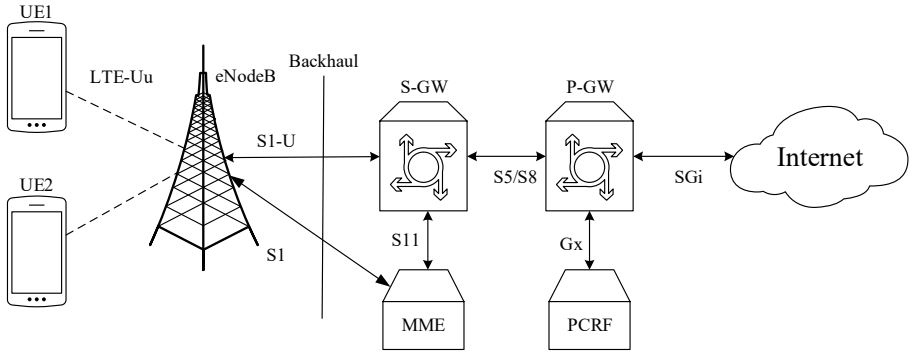



Figure 8.1: The main elements of the LTE network, relevant to the work presented in the thesis.

LTE networks can be divided into two main parts: the Evolved-Universal Terrestrial Radio Access Network (E-UTRAN) and Evolved Packet Core (EPC). The two parts of the LTE network are interconnected via the backhaul links, as shown in the above figure.

The E-UTRAN consists of two types of devices. The base-stations (also called Evolved Node Bs (eNodeBs)) are responsible for radio resources management, compression of user data and headers, user positioning and connectivity gateway functions between the User Equipment (UE) and the EPC. At the same time, UE is the user terminal, that implement the user side of radio access functions. In addition, modern UE devices can act as fully functioning computers with their own operating system and applications.

In the EPC, the main elements are as follows: the Mobility Management Entity (MME) is responsible for most of the control plane procedures. Among its tasks are support for the handover procedure (ensuring uninterrupted service when user roams between eNodeBs); tracking associations between UE and eNodeBs; data bearer management; and connections management.

While the MME handles the signaling, user-plane data is handled by the Serving Gateway (S-GW) and Packet Data Network Gateway (PDN-GW). The S-GW is used to collect charging data, Legal Interception (LI), and as a user-plane data anchor. The latter function is especially important in order to ensure uninterrupted user data transmission during user roaming between the eNodeBs.

Finally, the last covered element of EPC is the PDN-GW. It is responsible for IP addresses assignment to the UEs, QoS enforcement, and flow-based charging. The charging function in tje PDN-GW is implemented with the help of the Policy and Charging Enforcement Function (PCEF), which is used to store traffic flow templates and associated charging data.

The last important topic covering the basics of LTE network in the user-plane protocols stack, as shown in Figure 8.2. Here, protocols operating in the E-UTRAN are shown using shaded color and the remaining protocols operate in EPC. For the sake of brevity, the figure shows a single integrated S-GW/P-GW device instead of two individual devices. The protocol stack of an individual S-GW device would look identical to the integrated device's right-hand side, with the GTP-U being the topmost layer, since the S-GW device does not terminate the user's IP flows.

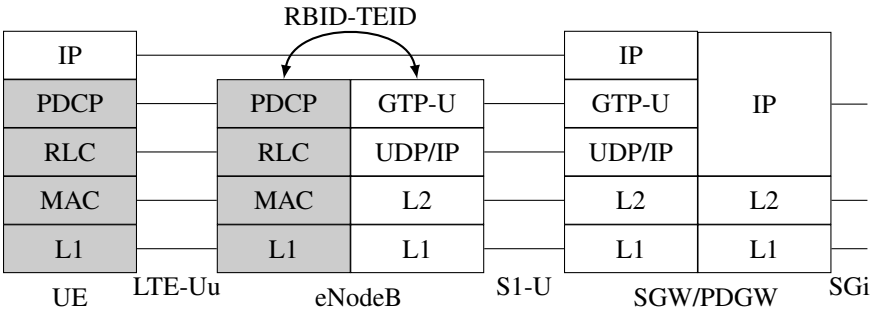



Figure 8.2: LTE userplane protocols stack.

For the work described later in the chapter, the most important part of the protocols stack is the user's IP layer data. It is important to note, that user IP packets are sourced by the UE and this data flow is terminated only in the PDN-GW. In the LTE-Uu interface, the user's IP data is encapsulated in the Packet Data Convergence Protocol (PDCP) packets, and each user is assigned an RBID identifier. In the the eNodeB, user's data is decapsulated from PDCP and is placed into GTP-U packets, without inspecting the actual data. Once data is placed in the GTP-U packets, each user is assigned a TEID identifier, used to label each individual data tunnel between the eNodeB the user is attached to and PDN-GW. In the networks with an individual S-GW device, the protocol stack of S-GW device is akin to that of eNodeB – specifically, the user data is not terminated in the S-GW.

## 8.2 Edge Caching in the Literature

The topic of caching content at the edges of the mobile network has been investigated from multiple perspectives. These works will be briefly presented in this section to provide a more comprehensive picture for the following work. In this context, it is common to refer to the concept of Content-Centric Mobile Networks (CCMNs) as caching becomes a prominent criteria in the network design process, intertwined with more typical ones, like communications-related and computing-related considerations, as pointed out by the authors of [153].

By equipping network elements with cache capabilities, two fundamental questions arise: which content should be cached and where. The former entails estimation of the popularity of the content where several solutions have been proposed, as, for example, the use of Extreme Learning Machine (ELM) in [154].

The optimal caching content placement has been investigated in [155]—among others—with a specific focus on proactive location selection based on UE mobility patterns.

More specifically, regarding the architecture of Mobile Edge Caching (MEC) systems, the authors of [156] point out that the role edge caching plays in mobile networks is ultimately the evolution of a need to progressively move the content closer to its consumer. As such, the use of CDNs as Core Network (CN) caches leads to the usage of BS caches in the Radio Access Network (RAN) in the context of Information-Centric Networking (ICN).

In diversified RAN architectures, such as Cloud-Radio Access Network (C-RAN), it becomes also relevant to exploit a further intermediate location for storage, i.e. the Baseband Unit (BBU) pool as noted in [156]. This is considered as a synthesis of the shortcomings of the other two types of locations. It is, however, possible to draw a comparison with similar works focused on simulation of a MEC deployment. The authors of [157], for example, present simulation results on the performances of a collaborative MEC algorithm in improving content access delay, among others.

The setup described in this chapter has been designed to be transparent to these different approaches of designing the RAN, so that it is possible to build on top of the works presented here to give insights into the framework of tools needed to deploy MEC solutions in both scenarios.

## 8.3 Edge Caching Proposal

An overview of the logical entities of the proposed edge-caching solution is displayed in Figure 8.3. In the proposed architecture, a content cache (e.g., a web-server serving web pages) is co-located with the eNodeBs. If a UE data request can be served from the cache, IP packets are routed toward the cache server instead of the S-GW. Implementation details on IP packets processing in the eNodeB are presented in Section 8.3.1.

Positioning the cache servers in the eNodeB changes the user data path, so that IP packets routed from the eNodeB toward the cache server no longer traverse the S-GW and the PDN-GW. Due to this change, it is important to consider what impact this has on the functions performed by those elements now omitted. The only affected function in the S-GW is LI, which can, however, be performed directly in the cache server. The traffic shaping performed by the PDN-GW on the Evolved Packet System (EPS) bearers is also affected. In order for the cache server to provide this service, communication with the Policy and Charging Rules Function (PCRF) is necessary via the Gx interface, as well as the implementation of the PCEF functionalities.

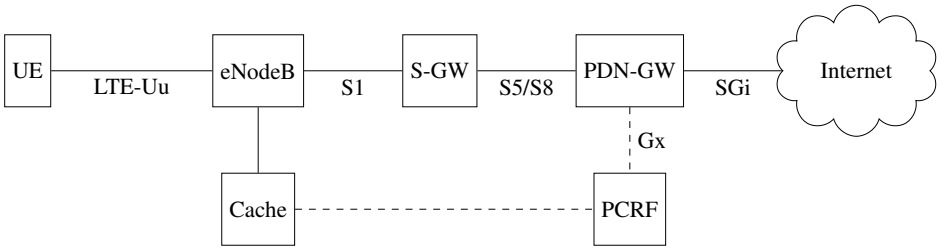



Figure 8.3: The logical architecture of the proposed edge-caching solution.

### 8.3.1 Protocol Stack

Starting from the standard user plane protocol stack described previously, a number of modifications are needed when equipping the eNodeB with a local cache. In the proposed MEC solution, the eNodeB has to not only act as a gateway, but also as a router. It has to inspect the traversing IP packets and, if the conditions are met, it has to divert the packets toward the cache server instead of the S-GW.

Moreover, a conventional eNodeB is transparent to the IP packets carried. However, to enable edge caching, this needs to change, as IP destination information is needed to properly route packets either locally to the cache or toward the EPC. In the proposed solution, the flow of the packets through a PDCP/GTP User (GTP-U) relay includes an additional step to inspect the destination IP address.

If the address is within a predefined range assigned to the cache servers, then the packet is sent to the local cache instead of the S-GW. In the proposed solution, the interface between the eNodeB and the local cache is also GTP-U-based. This minimizes the changes needed in the eNodeB, as from its perspective the interface to the local cache is like the interface to the S-GW. On the cache-server side, the GTP-U gateway in the testbed is implemented in the Linux Operating System (OS) kernel, allowing to use cache server integrating GTP-U and web-server functions (see Figure 8.4). Here, the the protocol layer implemented by the GTP-U gateway are shown in shaded grey, and the protocols for user data processing are shown with a white background.

Additionally, this MEC solution is IP-version agnostic and requires only a small extra processing time per packet in the eNodeB. To determine if a packet should be sent toward a local cache, the eNodeB performs checks on the IP version and destination address. A local cache can be made of several servers, but as long as all server addresses are within a single Classless Inter-Domain Routing (CIDR) block, it is enough to perform a single logical operation to determine where to route the packet.

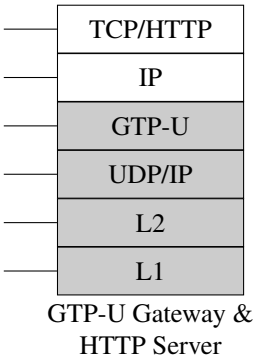



Figure 8.4: Protocol stack in the cache server with an integrated GTP-U gateway and HTTP server.

### 8.3.2 Content Location and Name Resolution

The previous section described the part of the caching solution that is responsible for data transport between the UE and the content cache. However, for it to work, the UE must know the IP address of a node that will fulfill the request. This address might be of a cache server, or of some other server if data is not available in the cache. The process of discovering the address of a network node that will serve the content request is part of the process of name resolution and routing and is described in this section.

Following ICN concepts, information that a UE tries to access, such as a web-page or a document, will be referred to as a Named Data Object (NDO) [158]. The discovery of NDO names is user-application specific and outside the scope of this work. However, once the NDO name is known, two possible methods for resolving it to an IP address are described in the following.

For NDOs identified using Uniform Resource Locators (URLs), a name resolution based on DNS can be used. In this case, a DNS server must be operated by the mobile operator. Upon reception of the client's DNS query, the server will perform two actions. First, it will query the MME to know the eNodeB that the client is attached to. Then, if the eNodeB in question contains a cache server, the cache server will be checked for the required content. If the cache server has the required content, the IP address of the cache server will be returned. Alternatively, the DNS server will perform a recursive DNS query and return the result to the client.

However, DNS-based NDO name resolution has several shortcomings. First, to optimize cache space usage, an operator might have different content available in different cache servers. This implies, that the NDO name resolution has to be performed by a DNS server having knowledge of all content available in edge-caches. This rules out cases where clients use different DNS servers than those deployed by the operator, as they do not contain such information. Second, to increase domain resolution speeds, the OS might

choose to cache the DNS replies. However, the cached address is valid only as long as the UE is connected to the same eNodeB and the content is still available in the edge-cache.

As an alternative to the DNS-based content location, NDO name can be resolved using the ALTO protocol [138]. To leverage ALTO for NDO IP address discovery, the ALTO server assigns a unique PID to each NDO. A PID is an alphanumeric identifier created by the operator to identify a network location. In this context, a network location is one or more IP addresses used by the cache servers containing the requested NDO. Upon receiving an ALTO protocol request to locate the PID, an ALTO server would perform the same two tasks as a DNS server, described previously. The results are then returned to the UE in the form of an ALTO *Network Map*. The returned network map contains a list of PIDs, each with an IP address of a cache server. Using the ALTO-provided network map to discover NDO IP addresses has several advantages. First, it allows the operator to inform the UE about all cache servers that contain the required NDO in single operation. Second, the UE can use additional ALTO protocol features (such as cost-maps) to inquire which cache-server is preferred in terms of data routing costs (i.e., available bandwidth, network load and others).

## 8.4 Performance Evaluation

In order to validate the proposed system, a prototype MEC implementation was evaluated for correctness and performance. The goals of the evaluation were two-fold. First, the tests were used to prove that the testbed worked as intended. This was achieved by ensuring that IP packets were routed in the eNodeB toward the cache when required. Second, a performance comparison was performed to quantify the improvements in terms of parameters impacting the user-perceived Quality of Experience (QoE). The overall network responsiveness was quantified by measuring the ping response times. On mobile devices, about 20% of all data traffic is used for web-browsing[159] delivered over the Hypertext Transfer Protocol (HTTP) protocol. The improvement in web-browsing performance was measured by observing the HTTP response rate. Finally, about 60% of all mobile data traffic is currently used to deliver streaming audio and video[159]. The improvement in streaming multimedia performance was measured by observing the average UDP packets jitter. The performance of the proposed system in ping and HTTP tests was compared to that of testing against cloud and CDN servers—two common methods of serving and also caching data in the Internet.

### 8.4.1 Performance Metrics Definition

The performance of the system prototype was tested using the following configuration. All tests were carried out using the reference implementation shown in Figure 8.5. The S1 interface link delay between the eNodeB and the EPC was set to 5 ms, which is considered as a realistic estimate of the delay over the S1 interface of a real network. A computer running Linux Ubuntu 16.04 with a Huawei E3372 LTE dongle was used as the UE. For

over-the-air testing, the LTE-Uu interface was set to a bandwidth of 10Mhz (50 Resource Blocks).

Ping tests were performed by sending 100 consecutive *ping* requests between the computer with the Commercial Off-The-Shelf (COTS) LTE modem (UE, node 1 in Figure 8.5) and a cache server (node 3 in Figure 8.5) connected directly to the eNodeB (node 2 in Figure 8.5), a server in a public cloud (node 5 in Figure 8.5), and a CDN server (node 6 in Figure 8.5). The same test was also repeated from the EPC (node 4 in Figure 8.5); however, without pinging the cache server co-located with the eNodeB.

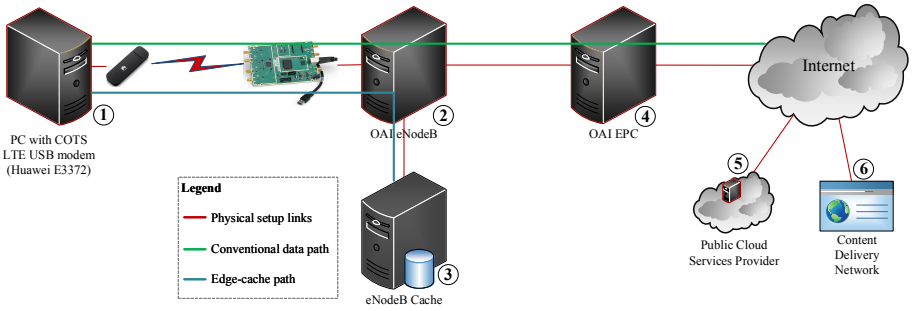



Figure 8.5: Physical components used to implement the edge-caching system prototype.

The HTTP performance tests were carried out using the *httperf* software. During the tests, a web server distributing a 500 KB file was used. The size of the payload was derived by taking the average size of Hypertext Markup Language (HTML), JavaScript and Cascading Style Sheets (CSS) content as observed among the 100 most popular Internet web-sites [160]. Each test consisted of 1500 requests for such a file, with a new TCP connection created for each request. The concurrent test load in each case was discovered by gradually increasing the number of concurrent connections, until the client connections started to time-out. The discovered value was later used to repeat the whole test. The indicated test results show the average number of responses per second as observed in 5-second measurement windows. Performing 1500 requests allowed to collect enough 5-second measurement windows to derive statistically valid results.

The UDP packet jitter tests were carried out using *iPerf3* software. The offered data load was 25 Mbps in order to fully saturate the connection. The actual jitter calculation follows the description in [161]. As this test requires test software to be run on both connection ends, jitter tests were not carried out between the UE and the CDN.

The results presented here were obtained using a single UE connected to the eNodeB without any traffic shaping in effect. The performance in the real mobile networks will depend, among other parameters, on the number of UEs connected to the same eNodeB,

on the scheduling algorithm in the eNodeB, on the traffic shaping parameters and on the channel conditions in the LTE-Uu interface among other factors.

### 8.4.2 Results and Discussion

The evaluation of the MEC solution focused on both artificial and real-world usage tests. The results of the evaluation is shown in Figure 8.6.

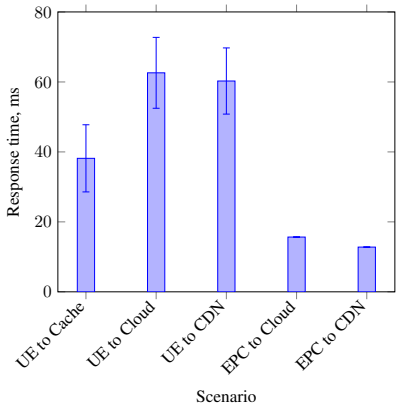
Figure 8.6a shows the average ping time from the UE to the edge-cache server, a server hosted in the public cloud and a server in the CDN. Ping measurements from the EPC to the servers in the public cloud and the CDN are included to have a reference delay once packets traverse the Fronthaul (FHL) and Backhaul (BHL) networks. Using the prototype implementation, the ping time from the UE to the edge-cache is 39% (24.45 ms) and 37% (22.11 ms) shorter compared to the ping time to the cloud and CDN servers. These results can be adjusted to remove the influence of the BHL delay (5 ms) and the delay from the EPC to the remote servers. By taking these adjustments, the time savings from the UE to the server in the public cloud is 10% (3.78 ms) and 11% (4.32 ms) to the CDN server.

Figure 8.6b shows the average number of HTTP responses received per second from the edge-cache and the servers in the cloud and in the CDN. This represents a real-life test, as an increase in the number of HTTP responses per time window leads to a shorter application response time, which ultimately improves the perceived web-browsing QoE. The results indicate that by serving content from the edge-cache, the response rate can be increased 3.6 and 1.73 times, compared to the cloud and CDN servers, respectively. The figure also shows that the response rate in the edge-cache case approaches that of accessing remote servers from the EPC. A big difference in the performance can partly be explained by the shorter response times between the UE and the edge server compared to Cloud and CDN use-cases. As TCP connection establishment, tear-down, and data transfer require multiple data round-trips, a shorter delay to the edge-cache server corresponds to a higher overall response rate.

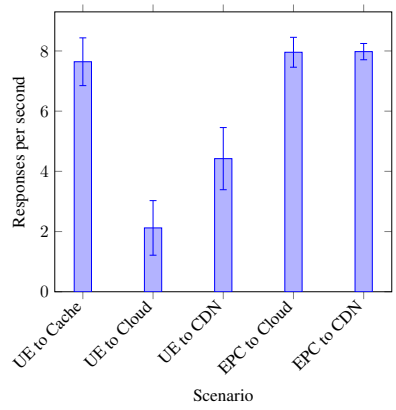
The quality of real-time streaming content is affected by the average packet jitter [161]. Figure 8.6c shows the average observed jitter from the UE to the edge-cache and the server in the cloud. As shown in Figure 8.6c, the use of edge-cache provides a marginal improvement of 6% (0.42 ms). This also shows that most of the jitter is then induced on the LTE-Uu interface between the UE and the eNodeB.

## 8.5 Extending Edge-caching Solution for P2P Communication

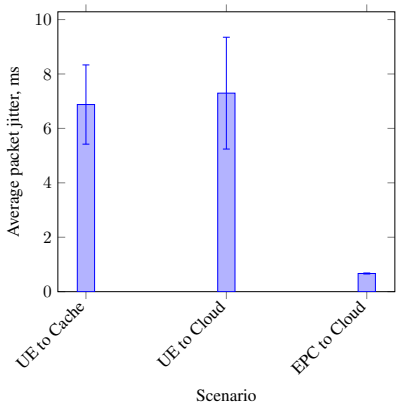
The architecture and protocol stack of LTE network are not designed for P2P communication. This mainly stems from the fact, that all user-plane data is encapsulated into GTP-U tunnels at eNodeBs and then sent encapsulated to PDN-GW (c.f. Figure 8.2). Here the destination IP address of user packets are inspected and packets are routed accordingly. Even if the packet is addressed to the user connected to the same eNodeB as a sender, the




(a) Average ping response time




(b) Average number of HTTP responses




(c) Average UDP packet jitter

Figure 8.6: Test results of prototype edge-caching solution. UE - User equipment, EPC - Evolved Packet Core, CDN - Content Delivery Network. Error bars indicate one standard deviation.

packet must traverse the whole network twice – once on the way to the PDN-GW and the second way back.

In order to simplify P2P communication in the LTE network, the prototype edge-caching setup was upgraded to allow IP packets routing at the eNodeB. The following section describes the proposed P2P communication solution. In order to help with the description, Figure 8.7 is used for the reference.

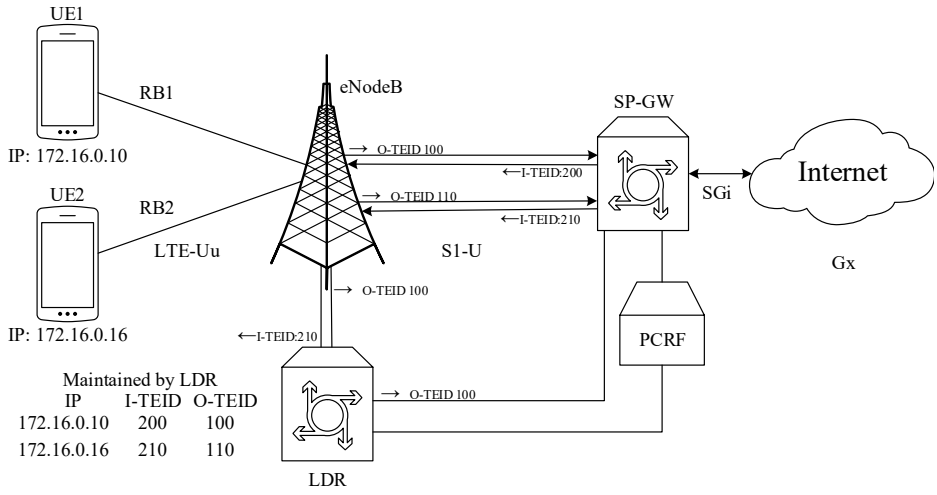



Figure 8.7: Components of LTE based P2P communication system

The figure shows two UEs connected to the eNodeB. UE1 is assigned IP address `172.16.0.10` and UE2 is assigned IP address `172.10.0.16`. Additionally, the figure shows an integrated S-GW and PDN-GW device, PCRF, and an additional device called Local Data Router (LDR), described later.

The eNodeB and SP-GW are shown connected by four lines, each indicating a one-way GTP-U tunnel and identified by TEID number. Data from UE1 is transported in tunnels having TEIDs 100 and 200, and data from UE2 in tunnels having TEIDs 110 and 210.

The eNodeB in the proposed solution functions as in the edge-caching setup. Namely, before encapsulating IP packets received over the air interface, it checks the destination IP address. If the address is in the range of IP addresses assigned to the UEs, it encapsulates the packet as normal; however, instead of sending it to SPGW, it sends it to LDR. As in the edge-caching use-case, checking if the IP address is in the subnet range can be performed using a single bitwise-logic operation.

In the proposed solution, the LDR functions as a data router. It decapsulates all received packets from the eNodeB and inspects the destination IP address. If the destination IP address is assigned to another UE attached to the same eNodeB, it encapsulates the packet using the correct TEID and sends it back to the eNodeB. Otherwise, the packet is sent as received (encapsulated in GTP-U) to the SP-GW.

Using the setup shown in the figure above as an example, all data received by the LDR sent by the UE1 will be encapsulated in the GTP-U tunnel identified by TEID 200. After the received data is decapsulated, the destination IP is inspected by the LDR. If the destination is UE2, the data is encapsulated in the GTP-U tunnel with TEID 210 and sent back to the eNodeB, where it is processed as if it was received from SP-GW. If the data is not addressed to the UE2, LDR encapsulates data into the GTP-U tunnel with TEID 100 and forwards data to SP-GW. Here, data is processed as if it was sent by the eNodeB.

Here it is important to note that LDR "impersonates" both eNodeB (when sending data to SP-GW), and SP-GW (when sending data to eNodeB). This is made possible by the fact that GTP-U is sent using a connectionless UDP protocol. If required, even the source IP address of LDR can be overwritten to that of eNodeB or SP-GW when sending "impersonated" data. This way, data sent (for example) from LDR to SP-GW is indistinguishable from the data sent from eNodeB to SP-GW.

From the description above, it can be seen that in order for the proposed solution to work correctly, the LDR must know the TEID identifiers associated with each UE connected to the eNodeB. This information can be obtained either from the MME or from the eNodeB. The prototype implementation obtains the TEIDs from eNodeB using the following procedure. When the UE attaches to the eNodeB, it initiates the default bearer setup procedure. During this procedure, eNodeB and MME exchanges `S1AP Initial Context Setup Request` and `Innitial Context Setup Response` messages. These messages contain TEIDs for both up-link and down-link tunnels. Upon reception of both messages in the eNodeB, the code running in the eNodeB forwards these TEIDs to the LDR.

Finally, it is important to note that user-plane P2P data sent via LDR no longer traverses the PDN-GW. This implies, that the traffic shaping must be done by the LDR. For this reason, a Gx interface between the LDR and PCRF is envisioned (although not implemented in the prototype).

## 8.6 P2P Solution Experimental Evaluation

The described P2P communication over the LTE network solution was implemented using an edge-caching test-bed. For it to work, two changes were made to the set-up: the eNodeB program code was augmented with TEIDs tracking and LDR notification code; and the web server software was removed from the edge-caching server and IP routing turned on. The IP packets routing function in the LDR server was implemented using a conventional Linux Kernel IP routing functionality.

The testing of the P2P communication solution was envisioned to follow the same process as for the edge-caching setup. However, the scheduler used in the OpenAirInterface eNodeB software does not work in a stable mode when more than one UE is connected. This is an (now) known issue with the scheduler code, causing eNodeB code to crash once the data load to the UEs increases. This was not an issue in the edge-caching setup, as only one UE was connected at the same time.

Due to the above reason, the P2P communication solution was tested using the *ping* program, as ping packets are small enough to not crash the eNodeB. The performance of the proposed system prototype was tested using the following setup. A Huawei E3372[162] USB dongle and a Cisco 819[163] D2D router with a Sierra Wireless MC7300 series LTE modem acted as UEs. The USB dongle was connected directly to the PC, while the Cisco router was connected to the PC over a GigabitEthernet link. Both UEs were placed approximately 50 centimeters from the SDR functioning as eNodeB. The S1 interface link delay between the eNodeB and the EPC was set to 5 ms, which is considered as a realistic estimate of a real LTE network S1 interface delay.

The results of the testing are shown in Table 8.1. The results indicate that the ping time via the LDR is 15.08 ms shorter than via the EPC. Of this time, 10 ms should be attributed to the S1 interface round-trip delay. This shows that by using LDR, communication delay between the peers connected to the same eNodeB can be shortened by 5.08 ms. Furthermore, the delay is less varying. Standard deviation of delay measurements is 2.1 times lower when using LDR compared to measurements of EPC routed data. As delay variation (jitter) has a negative impact on the quality of real-time communication, lower delay variation can lead to increase of P2P communication QoS.

Table 8.1: LTE P2P communication test-bed testing results

| Scenario | Ping time, ms | Std. dev. |
|---|---|---|
| Via Local Data Router | 62.28 | 9.28 |
| Via Evolved Packet Core | 77.36 | 19.54 |

## 8.7 Conclusions

MEC solutions propose the use of content caches located in the base stations (eNodeBs) to serve popular content to the UEs. Co-locating the cache server with the eNodeB offers several advantages. Besides improving QoE for the UEs, it also reduces the load on the operators' BHL network. A prototype of a MEC system has been presented in this chapter. It focuses on minimizing the changes needed in order to implement it in a real network. In order to verify the design of the proposed edge-caching system, the prototype has been implemented and tested using a COTS UE.

The same test-bed was later improved to enable P2P communication over the LTE networks. While the testing of the P2P communication suffered from the poor quality of the scheduler algorithm in the eNodeB software, it was still shown that the solution works and the delay jitter can be reduced by using such a solution.

The goal of the work presented in this chapter was to demonstrate a practical feasibility of the mobile edge-caching system and the related P2P communication system. While the prototype did achieve the initial goal, there are still a number of open questions for future research. Among them, are the questions relating to the performance of such systems when a number of users is high and feasibility of performing multimedia streaming and P2P communication between the base stations (over the X2 interface).

CHAPTER 9

# Work Summary and Future Outlook

An unprecedented growth of Internet video traffic puts a strain on the Internet Services Provider (ISP) and CDN operators. As the amount of Internet video is expected to continue growing, new methods of streaming video delivery are being actively researched. One of the possible methods to increase the number of users that can receive streaming multimedia while utilizing the already-deployed multimedia streaming infrastructure (streaming servers) is P2P-assisted multimedia streaming.

A client performing P2P-assisted multimedia streaming works like a conventional streaming client–it receives streaming media from the streaming server and renders it to the user. However, as it receives data from the actual streaming server, it starts acting as a streaming server itself. It advertises the presence of multimedia data to other clients and responds to data requests from other clients. By dong so, it allows to reduce the communication load on the actual streaming server, hence increasing the number of users one streaming server can serve. Experimental deployment of such solutions for CDN offload are already underway by at least one major CDN provider.

This work provides a comprehensive overview of the proposed P2P communication protocols, with extra focus on the protocols designed for multimedia streaming. A description of different types of protocols is provided along with analysis of the different operation states of the P2P client. A comprehensive description of the PPSPP client, used throughout the thesis, is given. This is supplemented with the description of the methods used to quantify the QoS of multimedia streaming.

As described before, during P2P-assisted streaming clients request multimedia data from the streaming server and from other clients. This thesis proposes a data-requesting algorithm for use in the PyPPSPP client when streaming multimedia. The impact of the algorithm's configuration parameters on the streaming multimedia's QoS is analyzed. The main outcomes of this analysis indicate that the best results are reached when the amount of requested data is small. The results indicate that the highest quality levels are achieved when the amount of requested data is up to 200 data chunks.

The PPSPP client utilized in this thesis used LEDBAT for congestion control. When using LEDBAT, data flows originating from the P2P client yields to other data flows in the congested network link. This is done to minimize the impact of P2P data transfer on other data flows. This, however, is sub-optimal for multimedia streaming. This work takes the

position, that a user performing multimedia streaming would rather have an uninterrupted multimedia stream at the bandwidth expense of other data flows. To achieve this, a set of configuration parameters for LEDBAT is proposed. The results indicate, that the start-up time is decreased and the playback continuity is increased when the proposed LEDBAT configuration is used.

The QoS of received multimedia depends on the selection of the communication peers. The possibility of using ALTO to influence the selection of peers is investigated. By using ALTO, each P2P client ranks the peers based on the data-routing cost between itself and the remote client, as provided by the ALTO server. This work proposes several methods that can be used to derive this data-routing cost value in the ALTO server. The experiment results show that the data-routing cost, derived from the routing protocol metrics, works best in the test network topologies used.

This work concludes with an analysis of P2P communication over two types of wireless communication protocols–Wi-Fi Peer-to-Peer and LTE. In order to enable P2P multimedia streaming over the Wi-Fi P2P connections, this work proposes a number of changes to the PPSPP protocols. The performance of multimedia streaming over the Wi-Fi P2P is analyzed using a use-case of multimedia streaming in train. The results indicate that such a use-case is feasible only for a small number of concurrent streaming clients.

This work also analyses the possible methods of adapting LTE networks for higher-quality multimedia delivery. This work proposes to reduce the latency and jitter in multimedia delivery by placing streaming multimedia caches (streaming servers) in the mobile base-stations. A proof-of-concept implementation of such a proposal is described and its performance analyzed. The results show that such deployment is indeed feasible and does improve the communication parameters (delay, jitter).

### 9.0.1 Future Work

While this thesis looked into a wide array of processes happening in the P2P client, several questions remain open for future research.

The data requesting algorithm presented in this thesis was tested using only a single network topology. Further testing (preferably over the Internet) would be beneficial. Such testing should also be performed during different times of the day to experience varying background traffic levels. The testing should also include failing clients. The same applies to the further testing of the proposed LEDBAT changes.

The Wi-Fi P2P technology is a promising candidate for the implementation of short-range ad-hoc networks. However, most of the tests in this thesis were performed using a networks emulator. With the arrival of the latest version of Raspberry Pi computers with 2.4 and 5 GHz Wi-Fi P2P capabilities and an improved Wi-Fi antenna, tests should be re-run using real devices. These Wi-Fi P2P tests should also be used to tune LEDBAT for operation over Wi-Fi P2P connections.

Finally, the upcoming 5G mobile networks will most likely be a key ingredient in device-to-device (D2D) Internet-of-Things (IoT) communication. It remains to be seen what is the best method to implement D2D communication in 5G networks. Different communication profiles (low-bandwidth sensors *vs.* HD security camera streaming) will require different strategies of data delivery in the network. This will provide ample opportunities for research and performance evaluation.

APPENDIX A

# PPSPP Messages

The following appendix describes the main message exchanges used in the PPSPP protocol.

## A.1 Handshake procedure

The handshake sequence is show in Figure A.1. Here, Peer A initiates the connection to Peer B.

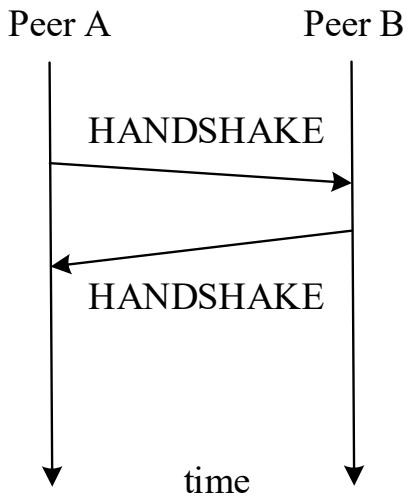



Figure A.1: PPSPP Handshake messages sequence.

The initiating HANDSHAKE message contains the following information: Local channel number; Remote Channel number (0); Version identifier; Swarm identifier; Content Integrity Protection method identifier; Merkle hash tree function; Live signature algorithm; Chunk addressing method; Live discard widow size; Supported messages

identifiers; Chunk size. The message was extended with the binary encoded UUID identifier.

The responding HANDSHAKE message is identical in its structure. The only difference, is that the Remote channel number is set to the value of the initiating message's local channel number.

## A.2 Data requesting and exchange

The process of data exchange is shown in Figure A.2.

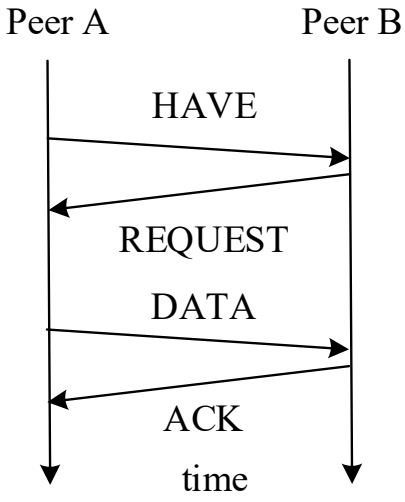



Figure A.2: PPSPP Data exchange sequence.

HAVE message contains data, identifying chunks present on the sending client. When a 32-bit chunk number are used to identify chunks, HAVE message contains the numbers identifying the first and last chunk number in a range. A client might send multiple HAVE messages to advertise multiple non-continuous chunk ranges.

REQUEST message has structure identical to the HAVE message.

DATA message contains the numbers identifying the first and the last chunk present in the DATA message's data field. In addition, DATA message contains a time-stamp when the message was sent.

ACK message contains the numbers identifying the first and the last chunk being acknowledged. The message also contains One-way delay samples for each data chunk acknowledged.

## A.3 Peers information requesting and exchange

The process of peers information exchange is shown in Figure A.2.

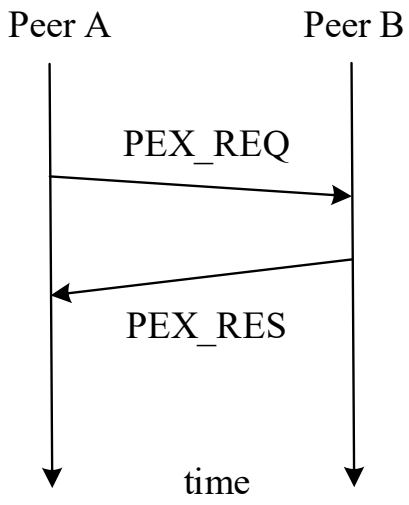



Figure A.3: PPSPP Peer information exchange sequence.

PEX_REQ message does not contain any additional information.

PEX_RES message contains the IP address and port number that some peer is using. It was extended to contain SSID networks identifier and the MAC address of the network.

APPENDIX B

# TCP Congestion Control Mechanism

TCP aims to limit congestion in communication links by maintaining a CWND – a number of bytes (or packets) that are sent into the networks but not yet acknowledged. Each sender of TCP data maintains its own CWND value. Figure B.1 shows a dynamic of CWND size over time.

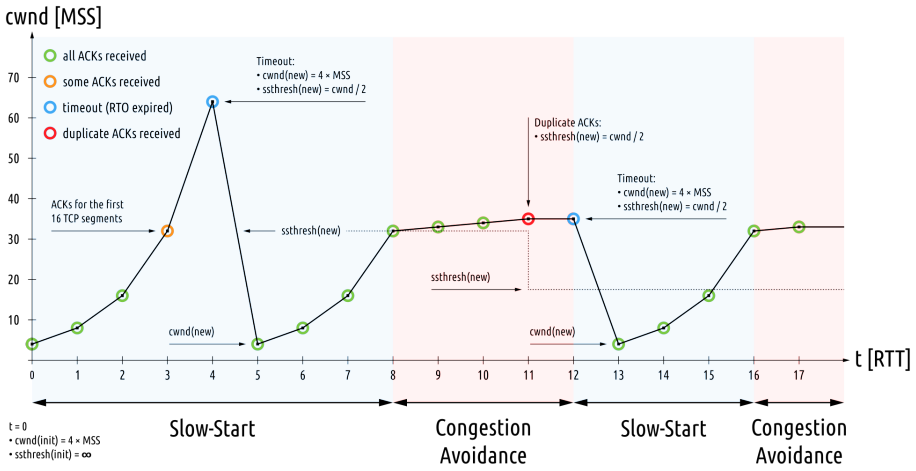



Figure B.1: Congestion window size of TCP protocols during various connection states. © GPLv3 Fleshgrinder / Wikimedia

Here, the vertical axis shows CWND expressed as multiples of Maximum Segment Size (MSS). During the slow-start phase, CWND size is increased by 1 for each ACK received. As the value of CWND grows, so grows the amount of data sent over the communication link. At a certain point (given that link has limited bandwidth), a buffer along the data path will overflow and discard some data. This will eventually result in a timeout event at the data sender, because ACK for the discarded data will not be received within the Retransmission Timeout (RTT) time. This indicates to the sender

that congestion occurred along the data path and the sending rate should be reduced. Reduction of the sending rate is done by reducing CWND to 4xMSS and setting the slow-start threshold to the half value of CWND before reduction.

Then, after an ACK loss and the reduction of CWND, the sender functions as already described, until the CWND value reaches a slow-start threshold. At this point the sender transitions into the congestion avoidance phase. In this phase, the CWND value is increased by 1 for each received ACK. As already described, a growing CWND value will eventually lead to buffer overflow along the data path and cause ACK to be missed. This will results reduction of CWND and transition back into the slow-start phase.

# Combined Bibliography


[1] *Cisco Visual Networking Index: Forecast and Methodology 2015-2020*. Tech. rep. 2016.

[2] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. “Deployment issues for the IP multicast service and architecture”. In: *IEEE network* 14.1 (2000), pp. 78–88.

[3] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. “Revisiting IP multicast”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 36. 4. ACM. 2006, pp. 15–26.

[4] Elliott Karpilovsky, Lee Breslau, Alexandre Gerber, and Subhabrata Sen. “Multicast redux: a first look at enterprise multicast traffic”. In: *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM. 2009, pp. 55–64.

[5] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. “The end-to-end effects of Internet path selection”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 29. 4. ACM. 1999, pp. 289–299.

[6] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. *Scalable application layer multicast*. Vol. 32. 4. ACM, 2002.

[7] Cristina L Abad, William Yurcik, and Roy H Campbell. “A survey and comparison of end-system overlay multicast solutions suitable for network-centric warfare”. In: *Proceedings of SPIE*. Vol. 5441. 2004, pp. 215–226.

[8] Xiangyang Zhang and Hossam Hassanein. “A survey of peer-to-peer live video streaming schemes–an algorithmic perspective”. In: *Computer Networks* 56.15 (2012), pp. 3548–3579.

[9] Meeyoung Cha, Pablo Rodriguez, Sue B Moon, and Jon Crowcroft. “On next-generation telco-managed P2P TV architectures.” In: *IPTPS*. 2008, p. 5.

[10] Gunnar Kreitz and Fredrik Niemela. “Spotify–large scale, low latency, P2P music-on-demand streaming”. In: *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*. IEEE. 2010, pp. 1–10.

[11] *The BitTorrent Protocol Specification*. `http://www.bittorrent.org/beps/bep_0003.html`. Version 0e08ddf84d8d3bf101cdf897fc312f2774588c9e. Accessed: 2017-09-01.

[12] Johan Pouwelse, Pawel Garbacki, Dick Epema, and Henk Sips. “The bittorrent p2p file-sharing system: Measurements and analysis”. In: *IPTPS*. Vol. 5. Springer. 2005, pp. 205–216.

[13] T. Hakala. *Windows Update for Business*. https://technet.microsoft.com/en-us/itpro/windows/plan/windows-update-for-business. Accessed: 2017-09-01. 2016.

[14] *3GPP - Releases*. http://www.3gpp.org/specifications/67-releases. Accessed: 2017-09-01.

[15] S.E. Deering. *Host extensions for IP multicasting*. RFC 1112 (Internet Standard). RFC. Updated by RFC 2236. Fremont, CA, USA: RFC Editor, Aug. 1989. DOI: 10.17487/RFC1112. URL: https://www.rfc-editor.org/rfc/rfc1112.txt.

[16] Pekka Savola. “Overview of the Internet multicast routing architecture”. In: (2008).

[17] H. Holbrook, B. Cain, and B. Haberman. *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast*. RFC 4604 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2006. DOI: 10.17487/RFC4604. URL: https://www.rfc-editor.org/rfc/rfc4604.txt.

[18] M. Christensen, K. Kimball, and F. Solensky. *Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches*. RFC 4541 (Informational). RFC. Fremont, CA, USA: RFC Editor, May 2006. DOI: 10.17487/RFC4541. URL: https://www.rfc-editor.org/rfc/rfc4541.txt.

[19] D. Waitzman, C. Partridge, and S.E. Deering. *Distance Vector Multicast Routing Protocol*. RFC 1075 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Nov. 1988. DOI: 10.17487/RFC1075. URL: https://www.rfc-editor.org/rfc/rfc1075.txt.

[20] J. Moy. *Multicast Extensions to OSPF*. RFC 1584 (Historic). RFC. Fremont, CA, USA: RFC Editor, Mar. 1994. DOI: 10.17487/RFC1584. URL: https://www.rfc-editor.org/rfc/rfc1584.txt.

[21] T. Bates, R. Chandra, D. Katz, and Y. Rekhter. *Multiprotocol Extensions for BGP-4*. RFC 2283 (Proposed Standard). RFC. Obsoleted by RFC 2858. Fremont, CA, USA: RFC Editor, Feb. 1998. DOI: 10.17487/RFC2283. URL: https://www.rfc-editor.org/rfc/rfc2283.txt.

[22] A. Adams, J. Nicholas, and W. Siadak. *Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)*. RFC 3973 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Jan. 2005. DOI: 10.17487/RFC3973. URL: https://www.rfc-editor.org/rfc/rfc3973.txt.

[23] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. *Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)*. RFC 4601 (Proposed Standard). RFC. Obsoleted by RFC 7761, updated by RFCs 5059, 5796, 6226. Fremont, CA, USA: RFC Editor, Aug. 2006. DOI: 10.17487/RFC4601. URL: https://www.rfc-editor.org/rfc/rfc4601.txt.

[24] D. Meyer, R. Rockell, and G. Shepherd. *Source-Specific Protocol Independent Multicast in 232/8*. RFC 4608 (Best Current Practice). RFC. Fremont, CA, USA: RFC Editor, Aug. 2006. DOI: 10.17487/RFC4608. URL: https://www.rfc-editor.org/rfc/rfc4608.txt.

[25] M. Handley, I. Kouvelas, T. Speakman, and L. Vicisano. *Bidirectional Protocol Independent Multicast (BIDIR-PIM)*. RFC 5015 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Oct. 2007. DOI: 10.17487/RFC5015. URL: https://www.rfc-editor.org/rfc/rfc5015.txt.

[26] Alexei Semenov. "Evolution of Peer-to-peer algorithms: Past, present and future." In: *Seminar on Internetworking, HUT T-110.551, Helsinki University of Technology*. 2005.

[27] Mojtaba Hosseini, Dewan Tanvir Ahmed, Shervin Shirmohammadi, and Nicolas D Georganas. "A survey of application-layer multicast protocols". In: *IEEE Communications Surveys & Tutorials* 9.3 (2007), pp. 58–74.

[28] Su-wei Tan, Gill Waters, and John Crawford. "A survey and performance evaluation of scalable tree-based application layer multicast protocols". In: (2003).

[29] John Jannotti, David K Gifford, Kirk L Johnson, M Frans Kaashoek, et al. "Overcast: reliable multicasting with on overlay network". In: *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation-Volume 4*. USENIX Association. 2000, p. 14.

[30] Paul Francis, Yuri Pryadkin, Pavlin Radoslavov, Ramesh Govindan, and Bob Lindell. *Yoid: Your own internet distribution*. 2000.

[31] Vincent Roca and Ayman El-Sayed. "A host-based multicast (HBM) solution for group communications". In: *Networking—ICN 2001* (2001), pp. 610–619.

[32] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. "ALMI: An application level multicast infrastructure". In: *USITS*. Vol. 1. 2001, pp. 5–5.

[33] Laurent Mathy, Roberto Canonico, and David Hutchison. "An overlay tree building control protocol". In: *Networked Group Communication* (2001), pp. 76–87.

[34] Sherlia Y Shi, Jonathan S Turner, and Marcel Waldvogel. "Dimensioning server access bandwidth and multicast routing in overlay networks". In: *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. ACM. 2001, pp. 83–91.

[35] Hrishikesh Deshpande, Mayank Bawa, and Hector Garcia-Molina. *Streaming live media over a peer-to-peer network*. Tech. rep. Stanford InfoLab, 2001.

[36] Venkata N Padmanabhan, Helen J Wang, Philip A Chou, and Kunwadee 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*. ACM. 2002, pp. 177–186.

[37] David A Helder and Sugih Jamin. "End-host multicast communication using switch-trees protocols". In: *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*. IEEE. 2002, pp. 419–419.

[38] Beichuan Zhang, Sugih Jamin, and Lixia Zhang. "Host multicast: A framework for delivering multicast to end users". In: *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 3. IEEE. 2002, pp. 1366–1375.

[39] Duc A Tran, Kien A Hua, and Tai T Do. "A peer-to-peer architecture for media streaming". In: *IEEE journal on Selected Areas in Communications* 22.1 (2004), pp. 121–133.

[40] Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, and Samir Khuller. "Construction of an efficient overlay multicast infrastructure for real-time applications". In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 2. IEEE. 2003, pp. 1521–1531.

[41] Jürgen Vogel, Jörg Widmer, Dirk Farin, Martin Mauve, and Wolfgang Effelsberg. "Priority-based distribution trees for application-level multicast". In: *Proceedings of the 2nd workshop on Network and system support for games*. ACM. 2003, pp. 148–157.

[42] Nodoka Mimura, Kiyohide Nakauchi, Hiroyuki Morikawa, and Tomonori Aoyama. "RelayCast: A middleware for application-level multicast services". In: *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*. IEEE. 2003, pp. 434–441.

[43] Zhichen Xu, Chunqiang Tang, Sujata Banerjee, and Sung-Ju Lee. "RITA: receiver initiated just-in-time tree adaptation for rich media distribution". In: *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. ACM. 2003, pp. 50–59.

[44] Yi Cui, Baochun Li, and Klara Nahrstedt. "oStream: asynchronous streaming multicast in application-layer overlay networks". In: *IEEE Journal on selected areas in communications* 22.1 (2004), pp. 91–106.

[45] Yong Zhong, Shervin Shirmohammadi, and AE Saddik. “Measurement of the effectiveness of application-layer multicasting”. In: *Instrumentation and Measurement Technology Conference, 2005. IMTC 2005. Proceedings of the IEEE*. Vol. 3. IEEE. 2005, pp. 2334–2339.

[46] Masahiro Kobayashi, Hidehisa Nakayama, Nirwan Ansari, and Nei Kato. “Robust and efficient stream delivery for application layer multicasting in heterogeneous networks”. In: *IEEE Transactions on Multimedia* 11.1 (2009), pp. 166–176.

[47] Vidhyashankar Venkataraman, Kaouru Yoshida, and Paul Francis. “Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast”. In: *Network Protocols, 2006. ICNP’06. Proceedings of the 2006 14th IEEE International Conference on*. IEEE. 2006, pp. 2–11.

[48] Beichuan Zhang, Wenjie Wang, Sugih Jamin, Daniel Massey, and Lixia Zhang. “Universal IP multicast delivery”. In: *Computer Networks* 50.6 (2006), pp. 781–806.

[49] Xing Jin, Kan-Leung Cheng, and S-H Gary Chan. “Island multicast: combining IP multicast with overlay data distribution”. In: *IEEE transactions on multimedia* 11.5 (2009), pp. 1024–1036.

[50] Andrea Magnetto, Rossano Gaeta, Marco Grangetto, and Matteo Sereno. “TURINstream: a totally push, robust, and efficient P2P video streaming architecture”. In: *IEEE Transactions on Multimedia* 12.8 (2010), pp. 901–914.

[51] Shabnam Ataee and Benoit Garbinato. “Eaglemacaw: A dual-tree replication protocol for efficient and reliable p2p media streaming”. In: *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. IEEE. 2014, pp. 112–121.

[52] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. “Application-level multicast using content-addressable networks”. In: *Networked Group Communication* (2001), pp. 14–29.

[53] Shelley Q Zhuang, Ben Y Zhao, Anthony D Joseph, Randy H Katz, and John D Kubiatowicz. “Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination”. In: *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. ACM. 2001, pp. 11–20.

[54] Jörg Liebeherr, Michael Nahas, and Weisheng Si. “Application-layer multicasting with delaunay triangulation overlays”. In: *IEEE Journal on Selected Areas in Communications* 20.8 (2002), pp. 1472–1488.

[55] Miguel Castro, Peter Druschel, A-M Kermarrec, and Antony IT Rowstron. “SCRIBE: A large-scale and decentralized application-level multicast infrastructure”. In: *IEEE Journal on Selected Areas in communications* 20.8 (2002), pp. 1489–1499.

[56] Rongmei Zhang and Y Charlie Hu. “Borg: a hybrid protocol for scalable application-level multicast in peer-to-peer networks”. In: *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. ACM. 2003, pp. 172–179.

[57] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. “Splitstream: High-bandwidth content distribution in cooperative environments”. In: *Peer-to-Peer Systems II* (2003), pp. 292–303.

[58] Ruixiong Tian, Yongqiang Xiong, Qian Zhang, Bo Li, Ben Y Zhao, and Xing Li. “Hybrid overlay structure based on random walks”. In: *IPTPS*. Vol. 5. Springer. 2005, pp. 152–162.

[59] Yatin Chawathe, Steven McCanne, and Eric A Brewer. “RMX: Reliable multicast for heterogeneous networks”. In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. IEEE. 2000, pp. 795–804.

[60] Yatin Dilip Chawathe and Eric A Brewer. *Scattercast: an architecture for internet broadcast distribution as an infrastructure service*. University of California, Berkeley, 2000.

[61] Sushant Jain, Ratul Mahajan, David Wetherall, Gaetano Borriello, and SD Gribble. *Scalable self-organizing overlays*. Tech. rep. Technical report UW-CSE 02-02-02, University of Washington, 2002.

[62] Mojtaba Hosseini and Nicolas D Georganas. “End system multicast protocol for collaborative virtual environments”. In: *Presence: Teleoperators and Virtual Environments* 13.3 (2004), pp. 263–278.

[63] Vinay Pai, Kapil Kumar, Karthik Tamilmani, Vinay Sambamurthy, and Alexander Mohr. “Chainsaw: Eliminating trees from overlay multicast”. In: *Peer-to-peer systems IV* (2005), pp. 127–140.

[64] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Y-SP Yum. “CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming”. In: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 3. IEEE. 2005, pp. 2102–2111.

[65] Li Zhao, Jian-Guang Luo, Meng Zhang, Wen-Jie Fu, Ji Luo, Yi-Fei Zhang, and Shi-Qiang Yang. “Gridmedia: A practical peer-to-peer based live video streaming system”. In: *Multimedia signal processing, 2005 IEEE 7th Workshop on*. IEEE. 2005, pp. 1–4.

[66] Susu Xie, Bo Li, Gabriel Y Keung, and Xinyan Zhang. “Coolstreaming: Design, theory, and practice”. In: *IEEE Transactions on multimedia* 9.8 (2007), pp. 1661–1671.

[67] Nazanin Magharei and Reza Rejaie. “Prime: Peer-to-peer receiver-driven mesh-based streaming”. In: *IEEE/ACM Transactions on Networking (TON)* 17.4 (2009), pp. 1052–1065.

[68] Mea Wang and Baochun Li. “R2: Random push with random network coding in live peer-to-peer streaming”. In: *IEEE Journal on Selected Areas in Communications* 25.9 (2007).

[69] Zhengye Liu, Yanming Shen, Keith W Ross, Shivendra S Panwar, and Yao Wang. “Substream trading: Towards an open P2P live streaming system”. In: *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*. IEEE. 2008, pp. 94–103.

[70] Dongni Ren, Yui-Tung Hillman Li, and S-H Gary Chan. “Fast-mesh: A low-delay high-bandwidth mesh for peer-to-peer live streaming”. In: *IEEE Transactions on Multimedia* 11.8 (2009), pp. 1446–1456.

[71] Zhengye Liu, Yanming Shen, Keith W Ross, Shivendra S Panwar, and Yao Wang. “LayerP2P: using layered video chunks in P2P live streaming”. In: *IEEE Transactions on Multimedia* 11.7 (2009), pp. 1340–1352.

[72] Feng Wang, Yongqiang Xiong, and Jiangchuan Liu. “mTreebone: A collaborative tree-mesh overlay network for multicast video streaming”. In: *IEEE Transactions on Parallel and Distributed Systems* 21.3 (2010), pp. 379–392.

[73] K-H Kelvin Chan, S-H Gary Chan, and Ali C Begen. “Spanc: Optimizing scheduling delay for peer-to-peer live streaming”. In: *IEEE Transactions on Multimedia* 12.7 (2010), pp. 743–753.

[74] Xiangyang Zhang and Hossam Hassanein. “Treeclimber: A network-driven push-pull hybrid scheme for peer-to-peer video live streaming”. In: *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE. 2010, pp. 368–371.

[75] Nazanin Magharei, Reza Rejaie, Ivica Rimac, Volker Hilt, and Markus Hofmann. “ISP-friendly live P2P streaming”. In: *IEEE/ACM Transactions on Networking (TON)* 22.1 (2014), pp. 244–256.

[76] Florian Rhinow, Pablo Porto Veloso, Carlos Puyelo, Stephen Barrett, and Eamonn O Nuallain. “P2P live video streaming in WebRTC”. In: *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*. IEEE. 2014, pp. 1–6.

[77] Matthias Wichtlhuber, Bjorn Richerzhagen, Julius Ruckert, and David Hausheer. “TRANSIT: Supporting transitions in Peer-to-Peer live video streaming”. In: *Networking Conference, 2014 IFIP*. IEEE. 2014, pp. 1–9.

[78] Mengjuan Liu, Xiaoshuan Ma, Xucheng Luo, Fei Lu, and Zhiguang Qin. “An ISP-friendly hierarchical overlay for P2P live streaming”. In: *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE. 2015, pp. 1–6.

[79] Nick Tindall and Aaron Harwood. “Peer-to-peer between browsers: cyclon protocol over WebRTC”. In: *Peer-to-Peer Computing (P2P), 2015 IEEE International Conference on*. IEEE. 2015, pp. 1–5.

[80] A. Bakker, R. Petrocco, and V. Grishchenko. *Peer-to-Peer Streaming Peer Protocol (PPSPP)*. RFC 7574. RFC. July 2015. DOI: 10.17487/RFC7574.

[81] Aggelos Vlavianos, Marios Iliofotou, and Michalis Faloutsos. “BiToS: Enhancing BitTorrent for supporting streaming applications”. In: *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*. IEEE. 2006, pp. 1–6.

[82] Thomas Bonald, Laurent Massoulié, Fabien Mathieu, Diego Perino, and Andrew Twigg. “Epidemic live streaming: optimal performance trade-offs”. In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 36. 1. ACM. 2008, pp. 325–336.

[83] Ralph Charles Merkle, Ralph Charles, et al. “Secrecy, authentication, and public key systems”. In: (1979).

[84] R. Cruz, M. Nunes, J. Xia, R. Huang, J. Taveira, and D. Lingli. *Peer-to-Peer Streaming Tracker Protocol (PPSTP)*. RFC 7846 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, May 2016. DOI: 10.17487/RFC7846. URL: https://www.rfc-editor.org/rfc/rfc7846.txt.

[85] Gojomo@bitzi.com. *MAGNET-URI Project*. http://magnet-uri.sourceforge.net. Accessed: 2017-09-01.

[86] JH Erik Andriessen. *Working with groupware: understanding and evaluating collaboration technology*. Springer Science & Business Media, 2012.

[87] Ayalvadi J Ganesh, A-M Kermarrec, and Laurent Massoulié. “Peer-to-peer membership management for gossip-based protocols”. In: *IEEE transactions on computers* 52.2 (2003), pp. 139–149.

[88] TS Eugene Ng and Hui Zhang. “Predicting Internet network distance with coordinates-based approaches”. In: *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 1. IEEE. 2002, pp. 170–179.

[89] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. “Vivaldi : A Decentralized Network Coordinate System”. In: *Communication* 34 (2004), pp. 15–26. ISSN: 01464833. DOI: 10.1145/1030194.1015471.

[90] Y. Rekhter and T. Li. *An Architecture for IP Address Allocation with CIDR*. RFC 1518 (Historic). RFC. Fremont, CA, USA: RFC Editor, Sept. 1993. DOI: 10.17487/RFC1518. URL: https://www.rfc-editor.org/rfc/rfc1518.txt.

[91] J. Postel. *User Datagram Protocol*. RFC 768 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/rfc/rfc768.txt.

[92] J. Postel. *Transmission Control Protocol*. RFC 793 (Internet Standard). RFC. Updated by RFCs 1122, 3168, 6093, 6528. Fremont, CA, USA: RFC Editor, Sept. 1981. DOI: 10.17487/RFC0793. URL: https://www.rfc-editor.org/rfc/rfc793.txt.

[93] S. Floyd, M. Handley, J. Padhye, and J. Widmer. *TCP Friendly Rate Control (TFRC): Protocol Specification*. RFC 5348 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Sept. 2008. DOI: 10.17487/RFC5348. URL: https://www.rfc-editor.org/rfc/rfc5348.txt.

[94] Reza Rejaie, Mark Handley, and Deborah Estrin. "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet". In: *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 3. IEEE. 1999, pp. 1337–1345.

[95] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. *Low Extra Delay Background Transport (LEDBAT)*. RFC 6817 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Dec. 2012. DOI: 10.17487/RFC6817. URL: https://www.rfc-editor.org/rfc/rfc6817.txt.

[96] Andrew Whitaker and David Wetherall. "Forwarding without loops in icarus". In: *Open Architectures and Network Programming Proceedings, 2002 IEEE*. IEEE. 2002, pp. 63–75.

[97] *How to Toggle Peer to Peer Protocol*. https://us.battle.net/support/en/article/how-to-toggle-peer-to-peer-protocol. Accessed: 2014-04-01.

[98] Riccardo Petrocco, Johan Pouwelse, and Dick HJ Epema. "Performance analysis of the Libswift P2P streaming protocol". In: *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*. IEEE. 2012, pp. 103–114.

[99] *UDP Tracker Protocol for BitTorrent*. http://www.bittorrent.org/beps/bep_0015.html. Accessed: 2017-09-01.

[100] M. Welzl and D. Ros. *A Survey of Lower-than-Best-Effort Transport Protocols*. RFC 6297 (Informational). RFC. Fremont, CA, USA: RFC Editor, June 2011. DOI: 10.17487/RFC6297. URL: https://www.rfc-editor.org/rfc/rfc6297.txt.

[101] S. Floyd. *Congestion Control Principles*. RFC 2914 (Best Current Practice). RFC. Updated by RFC 7141. Fremont, CA, USA: RFC Editor, Sept. 2000. DOI: 10.17487/RFC2914. URL: https://www.rfc-editor.org/rfc/rfc2914.txt.

[102] Jacob Chakareski, Sangeun Han, and Bernd Girod. “Layered coding vs. multiple descriptions for video streaming over multiple paths”. In: *Multimedia Systems* 10.4 (2005), pp. 275–285.

[103] Yanjiao Chen, Kaishun Wu, and Qian Zhang. “From QoS to QoE: A tutorial on video quality assessment”. In: *IEEE Communications Surveys & Tutorials* 17.2 (2015), pp. 1126–1165.

[104] Mahesh Viswanathan and Madhubalan Viswanathan. “Measuring speech quality for text-to-speech systems: development and assessment of a modified mean opinion score (MOS) scale”. In: *Computer Speech & Language* 19.1 (2005), pp. 55–83.

[105] Shyamprasad Chikkerur, Vijay Sundaram, Martin Reisslein, and Lina J Karam. “Objective video quality assessment methods: A classification, review, and performance comparison”. In: *IEEE transactions on broadcasting* 57.2 (2011), pp. 165–182.

[106] Zhengfang Duanmu, Kai Zeng, Kede Ma, Abdul Rehman, and Zhou Wang. “A quality-of-experience index for streaming video”. In: *IEEE Journal of Selected Topics in Signal Processing* 11.1 (2017), pp. 154–166.

[107] Tobias Hoßfeld, Raimund Schatz, Ernst Biersack, and Louis Plissonneau. “Internet video delivery in youtube: from traffic measurements to quality of experience”. In: *Data Traffic Monitoring and Analysis*. Springer, 2013, pp. 264–301.

[108] Demostenes Z Rodriguez, Julia Abrahao, Dante C Begazo, Renata L Rosa, and Graca Bressan. “Quality metric to assess video streaming service over TCP considering temporal location of pauses”. In: *IEEE Transactions on Consumer Electronics* 58.3 (2012), pp. 985–992.

[109] Jingteng Xue, Dong-Qing Zhang, Heather Yu, and Chang Wen Chen. “Assessing quality of experience for adaptive HTTP video streaming”. In: *Multimedia and Expo Workshops (ICMEW), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1–6.

[110] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. “Measuring the quality of experience of HTTP video streaming”. In: *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE. 2011, pp. 485–492.

[111] Lawrence S Brakmo, Sean W O’Malley, and Larry L Peterson. *TCP Vegas: New techniques for congestion detection and avoidance*. Vol. 24. 4. ACM, 1994.

[112] Arun Venkataramani, Ravi Kokku, and Mike Dahlin. “TCP Nice: A mechanism for background transfers”. In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 329–343.

[113] Aleksandar Kuzmanovic and Edward W Knightly. “TCP-LP: low-priority service via end-point congestion control”. In: *IEEE/ACM Transactions on Networking (TON)* 14.4 (2006), pp. 739–752.

[114] Michele C Weigle, Kevin Jeffay, and F Donelson Smith. “Delay-based early congestion detection and adaptation in TCP: impact on web performance”. In: *Computer Communications* 28.8 (2005), pp. 837–850.

[115] Andrea De Vendictis, Andrea Baiocchi, and Michela Bonacci. “Analysis and enhancement of TCP Vegas congestion control in a mixed TCP Vegas and TCP Reno network scenario”. In: *Performance Evaluation* 53.3-4 (2003), pp. 225–253.

[116] David X Wei, Cheng Jin, Steven H Low, and Sanjay Hegde. “FAST TCP: motivation, architecture, algorithms, performance”. In: *IEEE/ACM transactions on Networking* 14.6 (2006), pp. 1246–1259.

[117] Yi-Cheng Chan, Chia-Liang Lin, Chia-Tai Chan, and Cheng-Yuan Ho. “CODE TCP: a competitive delay-based TCP”. In: *Computer Communications* 33.9 (2010), pp. 1013–1029.

[118] Shao Liu, Milan Vojnovic, and Dinan Gunawardena. “Competitive and considerate congestion control for bulk data transfers”. In: *Quality of Service, 2007 Fifteenth IEEE International Workshop on*. IEEE. 2007, pp. 1–9.

[119] *Windows Background Intelligent Transfer Service*. Accessed: 2016-01-01. URL: %5Curl%7Bhttp://msdn.microsoft.com/library/bb968799(VS.85).aspx%7D.

[120] *Apple LEDBAT implementation*. Accessed: 2018-01-01. URL: %5Curl%7Bhttps://opensource.apple.com/source/xnu/xnu-1699.22.81/bsd/netinet/tcp_ledbat.c%7D.

[121] *iPerf3*. Accessed: 2017-07-01. URL: https://iperf.fr (visited on 07/01/2017).

[122] Jeff Ahrenholz. “Comparison of CORE network emulation platforms”. In: *Proceedings - IEEE Military Communications Conference MILCOM* (2010), pp. 166–171. ISSN: 2155-7578. DOI: 10.1109/MILCOM.2010.5680218.

[123] *Network Configuration Example Configuring MX Series Universal Edge Routers for Service Convergence*. Juniper Networks, 2016.

[124] Sarthak Grover, Roya Ensafi, and Nick Feamster. “When the Rich Get Richer : A Case Study of Traffic Demand Response to Service-Plan Upgrades”. In: *International Conference on Passive and Active Network Measurement*. Springer International Publishing, 2016, pp. 124–135.

[125] Gregor Maier, Vern Paxson, U C Berkeley Icsi, and Mark Allman. “On Dominant Characteristics of Residential Broadband Internet Traffic”. In: *the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), pp. 90–102. DOI: 10.1145/1644893.1644904.

[126] Bram Cohen. “Incentives build robustness in BitTorrent”. In: *Workshop on Economics of Peer-to-Peer systems*. Vol. 6. 2003, pp. 68–72.

[127] Arnaud Legout, Nikitas Liogkas, Eddie Kohler, and Lixia Zhang. “Clustering and sharing incentives in bittorrent systems”. In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 35. 1. ACM. 2007, pp. 301–312.

[128] *libswift - The multiparty transport protocol.* Accessed: 2018-01-01. URL: %5Curl%7Bhttps://github.com/libswift/libswift%7D.

[129] John A Nelder and Roger Mead. “A simplex method for function minimization”. In: *The computer journal* 7.4 (1965), pp. 308–313.

[130] Bernard Wong, Aleksandrs Slivkins, and Emin Gün EG Emin Gun Sirer. “Meridian: A lightweight network location service without virtual coordinates”. In: *ACM SIGCOMM Computer Communication Review* 35.4 (2005), pp. 85–96. ISSN: 0146-4833. DOI: 10.1145/1090191.1080103.

[131] Jan Seedorf, Saverio Niccolini, Martin Stiemerling, Ettore Ferranti, and Rolf Winter. “Quantifying operational cost-savings through ALTO-guidance for P2P live streaming”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6236 LNCS (2010), pp. 14–26. ISSN: 03029743. DOI: 10.1007/978-3-642-15485-0_3.

[132] Liu Guanxiu, Ye Suqi, and Huang Xinli. “A Novel ALTO Scheme for BitTorrent-Like P2P File Sharing Systems”. In: *2013 Third International Conference on Intelligent System Design and Engineering Applications* (2013), pp. 135–139. DOI: 10.1109/ISDEA.2012.39.

[133] Danny Alex, Lachos Perez, Samuel Henrique, and Bucke Brito. “Delivering Application-Layer Traffic Optimization Services based on Public Routing Data at Internet eXchange Points”. In: ().

[134] Nattee Pinthong and Woraphon Lilakiatsakun. “Performance of BitTorrent-Like P2P File Sharing Systems inspired by ALTO”. In: *IEEE Region 10 Annual International Conference, Proceedings/TENCON* (2013). ISSN: 21593442. DOI: 10.1109/TENCON.2013.6719035.

[135] G. Malkin. *RIP Version 2*. RFC 2453. RFC. Nov. 1998. DOI: 10.17487/RFC2453.

[136] J. Moy. *OSPF Version 2*. RFC 2328. RFC. Apr. 1998. DOI: 10.17487/RFC2328.

[137] Manish Jain and Constantinos Dovrolis. “End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput”. In: *IEEE/ACM Transactions on Networking* 11.4 (2003), pp. 537–549. ISSN: 10636692. DOI: 10.1109/TNET.2003.815304. arXiv: arXiv:1011.1669v3.

[138] R. Alimi, R. Penno, Y. Yang, S. Kiesel, S. Previdi, W. Roome, S. Shalunov, and R. Woundy. *Application-Layer Traffic Optimization (ALTO) Protocol*. RFC 7285. RFC. Sept. 2014. DOI: 10.17487/RFC7285.

[139] D. Levi, P. Meyer, and B. Stewart. *Simple Network Management Protocol (SNMP) Applications*. RFC 3413. RFC. Dec. 2002. DOI: 10.17487/RFC3413.

[140] *Ericsson Mobility Report, June 2017*. Tech. rep. 2017. URL: ericsson.com/mobility-report.

[141] Marco Conti, Franca Delmastro, Giovanni Minutiello, and Roberta Paris. "Experimenting opportunistic networks with Wi-Fi Direct". In: *2013 IFIP Wireless Days (WD)*. IEEE, Nov. 2013, pp. 1–6. ISBN: 978-1-4799-0543-0. DOI: 10.1109/WD.2013.6686501. URL: http://ieeexplore.ieee.org/document/6686501/.

[142] *Wi-Fi Peer-to-Peer (P2P) Technical Specification Version 1.7*. Tech. rep. 2016, p. 201. URL: https://www.wi-fi.org/discover-wi-fi/specifications.

[143] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. "Device-to-device communications with Wi-Fi Direct: overview and experimentation". In: *IEEE wireless communications* 20.3 (2013), pp. 96–104.

[144] Paul J. Leach, Rich Salz, and Michael H. Mealling. *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122. July 2005. DOI: 10.17487/RFC4122. URL: https://rfc-editor.org/rfc/rfc4122.txt.

[145] Daniel T Fokum and Victor S Frost. "A survey on methods for broadband internet access on trains". In: *IEEE communications surveys & tutorials* 12.2 (2010), pp. 171–185.

[146] Hsin-Ta Chiao, Shih-Ying Chang, Kuan-Ming Li, Yi-Ting Kuo, and Ming-Chien Tseng. "Wi-Fi multicast streaming using AL-FEC inside the trains of high-speed rails". In: *Broadband Multimedia Systems and Broadcasting (BMSB), 2012 IEEE International Symposium on*. IEEE. 2012, pp. 1–6.

[147] Li Zhu, F Richard Yu, Bin Ning, and Tao Tang. "Cross-layer design for video transmissions in metro passenger information systems". In: *IEEE Transactions on Vehicular Technology* 60.3 (2011), pp. 1171–1181.

[148] Ouldooz Baghban Karimi, Jiangchuan Liu, and Chonggang Wang. "Seamless wireless connectivity for multimedia services in high speed trains". In: *IEEE Journal on selected areas in communications* 30.4 (2012), pp. 729–739.

[149] Shih-Ying Chang, Hsin-Ta Chiao, Xin-Yan Yeh, and Ming-Chien Tseng. "UDP-based file delivery mechanism for video streaming to high-speed trains". In: *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on*. IEEE. 2013, pp. 3568–3572.

[150] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. 3 Park Avenue, New York, NY 10016-5997, USA: IEEE, 2012.

[151] *WifiP2pConfig | Android Developers*. Accessed: 2017-10-01. URL: https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pConfig.html (visited on 10/01/2017).

[152] Jaber Kakar and Gherekhloo. “Fundamental limits on latency in cloud- and cache-aided HetNets”. eng. In: *2017 Ieee International Conference on Communications (icc). Proceedings* (2017). Ed. by David Gesbert, 6 pp., 6 pp. ISSN: 19381883, 15503607. DOI: 10.1109/ICC.2017.7996346.

[153] Sergey Andreev, Olga Galinina, Alexander Pyattaev, Jiri Hosek, Pavel Masek, Halim Yanikomeroglu, and Yevgeni Koucheryavy. “Exploring Synergy between Communications, Caching, and Computing in 5G-Grade Deployments”. eng. In: *Ieee Communications Magazine* 54.8 (2016), pp. 60–69. ISSN: 15581896, 01636804. DOI: 10.1109/MCOM.2016.7537178.

[154] S. M. Shahrear Tanzil, William Hoiles, and Vikram Krishnamurthy. “Adaptive Scheme for Caching YouTube Content in a Cellular Network: Machine Learning Approach”. eng. In: *Ieee Access* 5 (2017), pp. 5870–5881. ISSN: 21693536. DOI: 10.1109/ACCESS.2017.2678990.

[155] Rui Wang, Xi Peng, Jun Zhang, and Khaled B. Letaief. “Mobility-Aware Caching for Content-Centric Wireless Networks: Modeling and Methodology”. eng. In: *Ieee Communications Magazine* 54.8 (2016), pp. 77–83. ISSN: 15581896, 01636804. DOI: 10.1109/MCOM.2016.7537180.

[156] Jianhua Tang and Tony Q. S. Quek. “The Role of Cloud Computing in Content-Centric Mobile Networking”. eng. In: *Ieee Communications Magazine* 54.8 (2016), pp. 52–59. ISSN: 15581896, 01636804. DOI: 10.1109/MCOM.2016.7537177.

[157] Tuyen X Tran, Parul Pandey, Abolfazl Hajisami, and Dario Pompili. “Collaborative multi-bitrate video caching and processing in mobile-edge computing networks”. In: *Wireless On-demand Network Systems and Services (WONS), 2017 13th Annual Conference on*. IEEE. 2017, pp. 165–172.

[158] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. “A survey of information-centric networking”. In: *IEEE Communications Magazine* 50.7 (2012).

[159] N. Heuveldop et. al. *Ericsson mobility report - June 2017*. Ericsson AB, 2017.

[160] *HTTP Archive - Trends*. Accessed: 2017-07-11. URL: http://httparchive.org/trends.php?s=Top100.

[161] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, July 2003. DOI: 10.17487/RFC3550. URL: https://www.rfc-editor.org/rfc/rfc3550.txt.

[162] *E3372 | Dongles | HUAWEI Global - HUAWEI Consumer*. https://consumer.huawei.com/en/mobile-broadband/e3372/specs/. Accessed: 2018-01-01.

[163] *Cisco 819 4G LTE M2M Gateway Integrated Service Routers Data Sheet*. https://www.cisco.com/c/en/us/products/collateral/routers/819-integrated-services-router-isr/data_sheet_c78-678459.html. Accessed: 2018-01-01.